Module 1 Summary

Module 1 Summary

Congratulations! You have completed this lesson. At this point in the course, you know that:

* Artificial intelligence (AI) simulates human cognition, while machine learning (ML) uses algorithms and requires feature engineering to learn from data.
* Machine learning includes different types of models: supervised learning, which uses labeled data to make predictions; unsupervised learning, which finds patterns in unlabeled data; and semi-supervised learning, which trains on a small subset of labeled data.
* Key factors for choosing a machine learning technique include the type of problem to be solved, the available data, available resources, and the desired outcome.
* Machine learning techniques include anomaly detection for identifying unusual cases like fraud, classification for categorizing new data, regression for predicting continuous values, and clustering for grouping similar data points without labels.
* Machine learning tools support pipelines with modules for data preprocessing, model building, evaluation, optimization, and deployment.
* R is commonly used in machine learning for statistical analysis and data exploration, while Python offers a vast array of libraries for different machine learning tasks. Other programming languages used in ML include Julia, Scala, Java, and JavaScript, each suited to specific applications like high-performance computing and web-based ML models.
* Data visualization tools such as Matplotlib and Seaborn create customizable plots, ggplot2 enables building graphics in layers, and Tableau provides interactive data dashboards.
* Python libraries commonly used in machine learning include NumPy for numerical computations, Pandas for data analysis and preparation, SciPy for scientific computing, and Scikit-learn for building traditional machine learning models.
* Deep learning frameworks such as TensorFlow, Keras, Theano, and PyTorch support the design, training, and testing of neural networks used in areas like computer vision and natural language processing.
* Computer vision tools enable applications like object detection, image classification, and facial recognition, while natural language processing (NLP) tools like NLTK, TextBlob, and Stanza facilitate text processing, sentiment analysis, and language parsing.
* Generative AI tools use artificial intelligence to create new content, including text, images, music, and other media, based on input data or prompts.
* Scikit-learn provides a range of functions, including classification, regression, clustering, data preprocessing, model evaluation, and exporting models for production use.
* The machine learning ecosystem includes a network of tools, frameworks, libraries, platforms, and processes that collectively support the development and management of machine learning models.

**Cheat Sheet: Linear and Logistic Regression**

**Comparing different regression types**

| **Model Name** | **Description** | **Code Syntax** |
| --- | --- | --- |
| Simple linear regression | **Purpose:** To predict a dependent variable based on one independent variable. **Pros:** Easy to implement, interpret, and efficient for small datasets. **Cons:** Not suitable for complex relationships; prone to underfitting. **Modeling equation:** y = b0 + b1x | 1. 1 2. 2 3. 3 4. from sklearn.linear\_model import LinearRegression 5. model = LinearRegression() 6. model.fit(X, y)   Copied!Wrap Toggled! |
| Polynomial regression | **Purpose:** To capture nonlinear relationships between variables. **Pros:** Better at fitting nonlinear data compared to linear regression. **Cons:** Prone to overfitting with high-degree polynomials. **Modeling equation:** y = b0 + b1x + b2x2 + ... | 1. 1 2. 2 3. 3 4. 4 5. 5 6. from sklearn.preprocessing import PolynomialFeatures 7. from sklearn.linear\_model import LinearRegression 8. poly = PolynomialFeatures(degree=2) 9. X\_poly = poly.fit\_transform(X) 10. model = LinearRegression().fit(X\_poly, y)   Copied!Wrap Toggled! |
| Multiple linear regression | **Purpose:** To predict a dependent variable based on multiple independent variables. **Pros:** Accounts for multiple factors influencing the outcome. **Cons:** Assumes a linear relationship between predictors and target. **Modeling equation:** y = b0 + b1x1 + b2x2 + ... | 1. 1 2. 2 3. 3 4. from sklearn.linear\_model import LinearRegression 5. model = LinearRegression() 6. model.fit(X, y)   Copied!Wrap Toggled! |
| Logistic regression | **Purpose:** To predict probabilities of categorical outcomes. **Pros:** Efficient for binary classification problems. **Cons:** Assumes a linear relationship between independent variables and log-odds. **Modeling equation:** log(p/(1-p)) = b0 + b1x1 + ... | 1. 1 2. 2 3. 3 4. from sklearn.linear\_model import LogisticRegression 5. model = LogisticRegression() 6. model.fit(X, y)   Copied!Wrap Toggled! |

**Associated functions commonly used**

| **Function/Method Name** | **Brief Description** | **Code Syntax** |
| --- | --- | --- |
| train\_test\_split | Splits the dataset into training and testing subsets to evaluate the model's performance. | 1. 1 2. 2 3. from sklearn.model\_selection import train\_test\_split 4. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)   Copied!Wrap Toggled! |
| StandardScaler | Standardizes features by removing the mean and scaling to unit variance. | 1. 1 2. 2 3. 3 4. from sklearn.preprocessing import StandardScaler 5. scaler = StandardScaler() 6. X\_scaled = scaler.fit\_transform(X)   Copied!Wrap Toggled! |
| log\_loss | Calculates the logarithmic loss, a performance metric for classification models. | 1. 1 2. 2 3. from sklearn.metrics import log\_loss 4. loss = log\_loss(y\_true, y\_pred\_proba)   Copied!Wrap Toggled! |
| mean\_absolute\_error | Calculates the mean absolute error between actual and predicted values. | 1. 1 2. 2 3. from sklearn.metrics import mean\_absolute\_error 4. mae = mean\_absolute\_error(y\_true, y\_pred)   Copied!Wrap Toggled! |
| mean\_squared\_error | Computes the mean squared error between actual and predicted values. | 1. 1 2. 2 3. from sklearn.metrics import mean\_squared\_error 4. mse = mean\_squared\_error(y\_true, y\_pred)   Copied!Wrap Toggled! |
| root\_mean\_squared\_error | Calculates the root mean squared error (RMSE), a commonly used metric for regression tasks. | 1. 1 2. 2 3. 3 4. from sklearn.metrics import mean\_squared\_error 5. import numpy as np 6. rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))   Copied!Wrap Toggled! |
| r2\_score | Computes the R-squared value, indicating how well the model explains the variability of the target variable. | 1. 1 2. 2 3. from sklearn.metrics import r2\_score 4. r2 = r2\_score(y\_true, y\_pred)   Copied!Wrap Toggled! |

**Author(s)**

Play

00:00

00:00

Mute

Settings

**Cheat Sheet: Building Supervised Learning Models**

**Common supervised learning models**

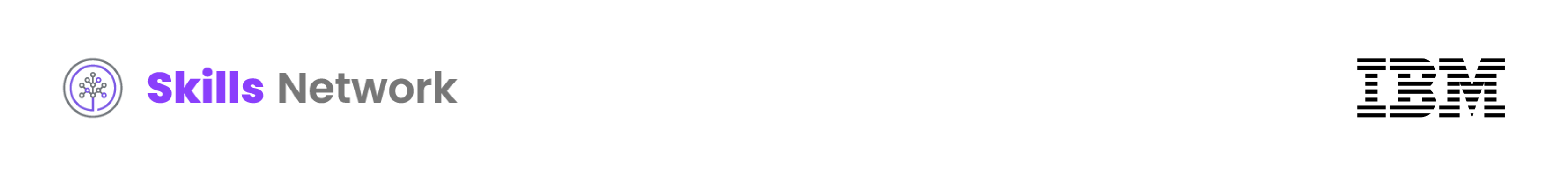
| **Process Name** | **Brief Description** | **Code Syntax** |
| --- | --- | --- |
| One vs One classifier (using logistic regression) | **Process:** This method trains one classifier for each pair of classes. **Key hyperparameters:** - `estimator`: Base classifier (e.g., logistic regression) **Pros:** Can work well for small datasets. **Cons:** Computationally expensive for large datasets. **Common applications:** Multiclass classification problems where the number of classes is relatively small. | 1. 1 2. 2 3. 3 4. from sklearn.multiclass import OneVsOneClassifier 5. from sklearn.linear\_model import LogisticRegression 6. model = OneVsOneClassifier(LogisticRegression())   Copied!Wrap Toggled! |
| One vs All classifier (using logistic regression) | **Process:** Trains one classifier per class, where each classifier distinguishes between one class and the rest. **Key hyperparameters:** - `estimator`: Base classifier (e.g., Logistic Regression) - `multi\_class`: Strategy to handle multiclass classification (`ovr`) **Pros:** Simpler and more scalable than One vs One. **Cons:** Less accurate for highly imbalanced classes. **Common applications:** Common in multiclass classification problems such as image classification. | 1. 1 2. 2 3. 3 4. from sklearn.multiclass import OneVsRestClassifier 5. from sklearn.linear\_model import LogisticRegression 6. model = OneVsRestClassifier(LogisticRegression())   Copied!Wrap Toggled!  or   1. 1 2. 2 3. from sklearn.linear\_model import LogisticRegression 4. model\_ova = LogisticRegression(multi\_class='ovr')   Copied!Wrap Toggled! |
| Decision tree classifier | **Process:** A tree-based classifier that splits data into smaller subsets based on feature values. **Key hyperparameters:** - `max\_depth`: Maximum depth of the tree **Pros:** Easy to interpret and visualize. **Cons:** Prone to overfitting if not pruned properly. **Common applications:** Classification tasks, such as credit risk assessment. | 1. 1 2. 2 3. from sklearn.tree import DecisionTreeClassifier 4. model = DecisionTreeClassifier(max\_depth=5)   Copied!Wrap Toggled! |
| Decision tree regressor | **Process:** Similar to the decision tree classifier, but used for regression tasks to predict continuous values. **Key hyperparameters:** - `max\_depth`: Maximum depth of the tree **Pros:** Easy to interpret, handles nonlinear data. **Cons:** Can overfit and perform poorly on noisy data. **Common applications:** Regression tasks, such as predicting housing prices. | 1. 1 2. 2 3. from sklearn.tree import DecisionTreeRegressor 4. model = DecisionTreeRegressor(max\_depth=5)   Copied!Wrap Toggled! |
| Linear SVM classifier | **Process:** A linear classifier that finds the optimal hyperplane separating classes with a maximum margin. **Key hyperparameters:** - `C`: Regularization parameter - `kernel`: Type of kernel function (`linear`, `poly`, `rbf`, etc.) - `gamma`: Kernel coefficient (only for `rbf`, `poly`, etc.) **Pros:** Effective for high-dimensional spaces. **Cons:** Not ideal for nonlinear problems without kernel tricks. **Common applications:** Text classification and image recognition. | 1. 1 2. 2 3. from sklearn.svm import SVC 4. model = SVC(kernel='linear', C=1.0)   Copied!Wrap Toggled! |
| K-nearest neighbors classifier | **Process:** Classifies data based on the majority class of its nearest neighbors. **Key hyperparameters:** - `n\_neighbors`: Number of neighbors to use - `weights`: Weight function used in prediction (`uniform` or `distance`) - `algorithm`: Algorithm used to compute the nearest neighbors (`auto`, `ball\_tree`, `kd\_tree`, `brute`) **Pros:** Simple and effective for small datasets. **Cons:** Computationally expensive as the dataset grows. **Common applications:** Recommendation systems, image recognition. | 1. 1 2. 2 3. from sklearn.neighbors import KNeighborsClassifier 4. model = KNeighborsClassifier(n\_neighbors=5, weights='uniform')   Copied!Wrap Toggled! |
| Random Forest regressor | **Process:** An ensemble method using multiple decision trees to improve accuracy and reduce overfitting. **Key hyperparameters:** - `n\_estimators`: Number of trees in the forest - `max\_depth`: Maximum depth of each tree **Pros:** Less prone to overfitting than individual decision trees. **Cons:** Model complexity increases with the number of trees. **Common applications:** Regression tasks such as predicting sales or stock prices. | 1. 1 2. 2 3. from sklearn.ensemble import RandomForestRegressor 4. model = RandomForestRegressor(n\_estimators=100, max\_depth=5)   Copied!Wrap Toggled! |
| XGBoost regressor | **Process:** A gradient boosting method that builds trees sequentially to correct errors from previous trees. **Key hyperparameters:** - `n\_estimators`: Number of boosting rounds - `learning\_rate`: Step size to improve accuracy - `max\_depth`: Maximum depth of each tree **Pros:** High accuracy and works well with large datasets. **Cons:** Computationally intensive, complex to tune. **Common applications:** Predictive modeling, especially in Kaggle competitions. | 1. 1 2. 2 3. import xgboost as xgb 4. model = xgb.XGBRegressor(n\_estimators=100, learning\_rate=0.1, max\_depth=5)   Copied!Wrap Toggled! |

**Associated functions used**

| **Method Name** | **Brief Description** | **Code Syntax** |
| --- | --- | --- |
| OneHotEncoder | Transforms categorical features into a one-hot encoded matrix. | 1. 1 2. 2 3. 3 4. from sklearn.preprocessing import OneHotEncoder 5. encoder = OneHotEncoder(sparse=False) 6. encoded\_data = encoder.fit\_transform(categorical\_data)   Copied!Wrap Toggled! |
| accuracy\_score | Computes the accuracy of a classifier by comparing predicted and true labels. | 1. 1 2. 2 3. from sklearn.metrics import accuracy\_score 4. accuracy = accuracy\_score(y\_true, y\_pred)   Copied!Wrap Toggled! |
| LabelEncoder | Encodes labels (target variable) into numeric format. | 1. 1 2. 2 3. 3 4. from sklearn.preprocessing import LabelEncoder 5. encoder = LabelEncoder() 6. encoded\_labels = encoder.fit\_transform(labels)   Copied!Wrap Toggled! |
| plot\_tree | Plots a decision tree model for visualization. | 1. 1 2. 2 3. from sklearn.tree import plot\_tree 4. plot\_tree(model, max\_depth=3, filled=True)   Copied!Wrap Toggled! |
| normalize | Scales each feature to have zero mean and unit variance (standardization). | 1. 1 2. 2 3. from sklearn.preprocessing import normalize 4. normalized\_data = normalize(data, norm='l2')   Copied!Wrap Toggled! |
| compute\_sample\_weight | Computes sample weights for imbalanced datasets. | 1. 1 2. 2 3. from sklearn.utils.class\_weight import compute\_sample\_weight 4. weights = compute\_sample\_weight(class\_weight='balanced', y=y)   Copied!Wrap Toggled! |
| roc\_auc\_score | Computes the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) for binary classification models. | 1. 1 2. 2 3. from sklearn.metrics import roc\_auc\_score 4. auc = roc\_auc\_score(y\_true, y\_score)   Copied!Wrap Toggled! |

**Author**

[Jeff Grossman](https://www.linkedin.com/in/jpgrossman/)  
[Abhishek Gagneja](https://www.linkedin.com/in/abhishek-gagneja-23051987/)



Play

00:00

00:00

Mute

Settings

**Cheat Sheet: Building Unsupervised Learning Models**

**Unsupervised learning models**

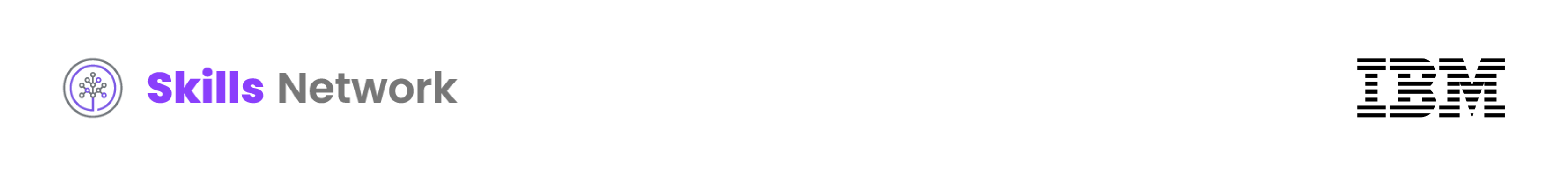
| **Model Name** | **Brief Description** | **Code Syntax** |
| --- | --- | --- |
| UMAP | UMAP (Uniform Manifold Approximation and Projection) is used for dimensionality reduction. **Pros:** High performance, preserves global structure. **Cons:** Sensitive to parameters. **Applications:** Data visualization, feature extraction. **Key hyperparameters:**   * **n\_neighbors:** Controls the local neighborhood size (default = 15). * **min\_dist:** Controls the minimum distance between points in the embedded space (default = 0.1). * **n\_components:** The dimensionality of the embedding (default = 2). | 1. 1 2. 2 3. from umap.umap\_ import UMAP 4. umap = UMAP(n\_neighbors=15, min\_dist=0.1, n\_components=2)   Copied!Wrap Toggled! |
| t-SNE | t-SNE (t-Distributed Stochastic Neighbor Embedding) is a nonlinear dimensionality reduction technique. **Pros:** Good for visualizing high-dimensional data. **Cons:** Computationally expensive, prone to overfitting. **Applications:** Data visualization, anomaly detection. **Key hyperparameters:**   * **n\_components:** The number of dimensions for the output (default = 2). * **perplexity:** Balances attention between local and global aspects of the data (default = 30). * **learning\_rate:** Controls the step size during optimization (default = 200). | 1. 1 2. 2 3. from sklearn.manifold import TSNE 4. tsne = TSNE(n\_components=2, perplexity=30, learning\_rate=200)   Copied!Wrap Toggled! |
| PCA | PCA (principal component analysis) is used for linear dimensionality reduction. **Pros:** Easy to interpret, reduces noise. **Cons:** Linear, may lose information in nonlinear data. **Applications:** Feature extraction, compression. **Key hyperparameters:**   * **n\_components:** Number of principal components to retain (default = 2). * **whiten:** Whether to scale the components (default = False). * **svd\_solver:** The algorithm to compute the components (default = 'auto'). | 1. 1 2. 2 3. from sklearn.decomposition import PCA 4. pca = PCA(n\_components=2)   Copied!Wrap Toggled! |
| DBSCAN | DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm. **Pros:** Identifies outliers, does not require the number of clusters. **Cons:** Difficult with varying density clusters. **Applications:** Anomaly detection, spatial data clustering. **Key hyperparameters:**   * **eps:** The maximum distance between two points to be considered neighbors (default = 0.5). * **min\_samples:** Minimum number of samples in a neighborhood to form a cluster (default = 5). | 1. 1 2. 2 3. from sklearn.cluster import DBSCAN 4. dbscan = DBSCAN(eps=0.5, min\_samples=5)   Copied!Wrap Toggled! |
| HDBSCAN | HDBSCAN (Hierarchical DBSCAN) improves on DBSCAN by handling varying density clusters. **Pros:** Better handling of varying densities. **Cons:** Can be slower than DBSCAN. **Applications:** Large datasets, complex clustering problems. **Key hyperparameters:**   * **min\_cluster\_size:** The minimum size of clusters (default = 5). * **min\_samples:** Minimum number of samples to form a cluster (default = 10). | 1. 1 2. 2 3. import hdbscan 4. clusterer = hdbscan.HDBSCAN(min\_cluster\_size=5)   Copied!Wrap Toggled! |
| K-Means clustering | K-Means is a centroid-based clustering algorithm that groups data into k clusters. **Pros:** Efficient, simple to implement. **Cons:** Sensitive to initial cluster centroids. **Applications:** Customer segmentation, pattern recognition. **Key hyperparameters:**   * **n\_clusters:** Number of clusters (default = 8). * **init:** Method for initializing the centroids ('k-means++' or 'random', default = 'k-means++'). * **n\_init:** Number of times the algorithm will run with different centroid seeds (default = 10). | 1. 1 2. 2 3. from sklearn.cluster import KMeans 4. kmeans = KMeans(n\_clusters=3)   Copied!Wrap Toggled! |

**Associated fuctions used**

| **Method** | **Brief Description** | **Code Syntax** |
| --- | --- | --- |
| make\_blobs | Generates isotropic Gaussian blobs for clustering. | 1. 1 2. 2 3. from sklearn.datasets import make\_blobs 4. X, y = make\_blobs(n\_samples=100, centers=2, random\_state=42)   Copied!Wrap Toggled! |
| multivariate\_normal | Generates samples from a multivariate normal distribution. | 1. 1 2. 2 3. from numpy.random import multivariate\_normal 4. samples = multivariate\_normal(mean=[0, 0], cov=[[1, 0], [0, 1]], size=100)   Copied!Wrap Toggled! |
| plotly.express.scatter\_3d | Creates a 3D scatter plot using Plotly Express. | 1. 1 2. 2 3. 3 4. import plotly.express as px 5. fig = px.scatter\_3d(df, x='x', y='y', z='z') 6. fig.show()   Copied!Wrap Toggled! |
| geopandas.GeoDataFrame | Creates a GeoDataFrame from a Pandas DataFrame. | 1. 1 2. 2 3. import geopandas as gpd 4. gdf = gpd.GeoDataFrame(df, geometry='geometry')   Copied!Wrap Toggled! |
| geopandas.to\_crs | Transforms the coordinate reference system of a GeoDataFrame. | 1. 1 2. gdf = gdf.to\_crs(epsg=3857)   Copied!Wrap Toggled! |
| contextily.add\_basemap | Adds a basemap to a GeoDataFrame plot for context. | 1. 1 2. 2 3. 3 4. import contextily as ctx 5. ax = gdf.plot(figsize=(10, 10)) 6. ctx.add\_basemap(ax)   Copied!Wrap Toggled! |
| pca.explained\_variance\_ratio\_ | Returns the proportion of variance explained by each principal component. | 1. 1 2. 2 3. 3 4. 4 5. from sklearn.decomposition import PCA 6. pca = PCA(n\_components=2) 7. pca.fit(X) 8. variance\_ratio = pca.explained\_variance\_ratio\_   Copied!Wrap Toggled! |

**Author**

[Jeff Grossman](https://www.linkedin.com/in/jpgrossman/)  
[Abhishek Gagneja](https://www.linkedin.com/in/abhishek-gagneja-23051987/)



Welcome to the final project, Rainfall Prediction Classifier. In this project, you will apply the knowledge and skills learned in this course to build a machine learning classifier that predicts rainfall based on historical weather data.

You will demonstrate your data science and machine learning skills by performing the following tasks, each of which includes specific steps:

1. Explore and prepare the dataset: Feature engineering and cleaning.
2. Build a classifier pipeline: Model selection, training, and optimization.
3. Evaluate the model’s performance: Interpret metrics and visualizations.

Imagine yourself as a data scientist at WeatherTech Inc., responsible for building a model that can predict whether or not it will rain tomorrow based on historical weather data. The dataset provided includes various weather features like temperature, humidity, and wind speed.

Your submission will be auto-graded using AI Mark.

<https://mark.skills.network/learner/771?lang=en>

Congratulations and Next Steps

Congratulations on completing this course! We hope you enjoyed the journey and feel equipped to leverage your new skills in the workplace or personal projects.

This course is part of the [IBM AI Engineering Professional Certificate](https://www.coursera.org/professional-certificates/ai-engineer) and [IBM Data Science Professional Certificate](https://www.coursera.org/enroll/ibm-data-science/paidmedia) programs, providing you with intermediate knowledge in applied machine learning. Throughout the course, you focused on essential machine learning techniques such as classification, regression, and clustering within supervised and unsupervised learning frameworks. Working with real-world data, you explored model development, assessment, and validation using Python and popular libraries, including Pandas, NumPy, and scikit-learn.

Additionally, you were introduced to advanced topics such as reinforcement learning and deep learning, setting the stage for future learning in AI. Hands-on labs and a final project offered practical experience, allowing you to solve data-driven problems and showcase your skills.

Whether you are kickstarting your career or building on existing knowledge, this course has equipped you with foundational machine learning skills relevant to career paths in Machine Learning, Data Science, and AI. We encourage you to continue exploring and applying these skills to advance in this exciting field.

**Next Steps**

As the next step, you are encouraged to explore the following courses if you are enrolled in the AI Engineering PC:

* [Introduction to Deep Learning & Neural Networks with Keras](https://www.coursera.org/learn/introduction-to-deep-learning-with-keras?specialization=ai-engineer)
* [Deep Learning with Keras and Tensorflow](https://www.coursera.org/learn/building-deep-learning-models-with-tensorflow?specialization=ai-engineer)
* [Introduction to Neural Networks and Pytorch](https://www.coursera.org/learn/deep-neural-networks-with-pytorch?specialization=ai-engineer)
* [Deep Learning with PyTorch](https://www.coursera.org/learn/advanced-deep-learning-with-pytorch?specialization=ai-engineer)
* [Generative AI and LLMs: Architecture and Data Preparation](https://www.coursera.org/learn/generative-ai-llm-architecture-data-preparation?specialization=ai-engineer)

If you are enrolled in the Data Science PC, you are encouraged to explore the following courses:

* [Generative AI: Elevate Your Data Science Career](https://www.coursera.org/enroll/learn/generative-ai-elevate-your-data-science-career/paidmedia?specialization=ibm-data-science)
* [Data Scientist Career Guide and Interview Preparation](https://www.coursera.org/enroll/learn/career-guide-and-interview-prep-for-data-science-pc/paidmedia?specialization=ibm-data-science)

We also encourage you to leave your feedback and rate this course so that we can continue to improve our content.

Good luck!