Setup Iniziale e Librerie

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Sklearn Preprocessing & Split
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Sklearn Models
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering

# Sklearn Metrics
from sklearn.metrics import mean_squared_error, r2_score, silhouette_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
    ConfusionMatrixDisplay
```

Listing 1: Tutti gli Import necessari

Il Preprocessing è la fase in cui trasformiamo i dati grezzi in un formato comprensibile per gli algoritmi di Machine Learning (che "ragionano" solo con i numeri).

1. Gestione dei Valori Mancanti (Missing Values)

Teoria: Gli algoritmi di ‘sklearn‘ non accettano `NaN` (Not a Number). Dobbiamo decidere se eliminare le righe (che però riduce il dataset) o riempirle (imputazione).

Requisito d’Esame: Spesso viene richiesto di imputare la media per i numeri e una stringa fissa per le categorie.

```python
from sklearn.impute import SimpleImputer

# Per le colonne numeriche: usiamo la media
# Strategy='mean' riempie i buchi con la media della colonna
num_imputer = SimpleImputer(strategy='mean')

# Per le colonne categoriche: usiamo un valore fisso
# Strategy='constant' mette un valore standard dove manca il dato
cat_imputer = SimpleImputer(strategy='constant', fill_value='unknown')
```

Listing 2: Imputazione dei dati

Approfondimento: Pandas vs Sklearn e il Data Leakage

Esiste un modo più veloce per riempire i valori mancanti usando direttamente Pandas, ma comporta dei rischi metodologici.

```python
# Metodo veloce: calcola la media su TUTTA la colonna e riempie i buchi
# Rischio: Se fatto prima dello split Train/Test, introduci Data Leakage!
df['colonna'] = df['colonna'].fillna(df['colonna'].mean())
```

Listing 3: Metodo rapido con Pandas (Attenzione al Leakage!)

Cos’è il Data Leakage?

Il **Data Leakage** (fuga di dati) avviene quando informazioni del *Test Set* (che il modello non dovrebbe mai vedere) influenzano il training.
Se calcoliamo la media su tutto il dataset prima di dividere:

1. La media usata per riempire i buchi nel Train Set contiene "tracce" dei valori del Test Set.

2. Il modello avrà performance ottimistiche in fase di training, ma peggiori su dati reali nuovi.

Procedura Corretta per l'Esame:

1. Dividere in `X_train` e `X_test`.
2. `imputer.fit(X_train)`: Impara la media solo dal Train.
3. `imputer.transform(X_train)` e `imputer.transform(X_test)`: Usa quella media per riempire entrambi.

2. Encoding (Da Testo a Numeri)

Teoria: I modelli matematici non sanno cosa sia "Rosso" o "Blu". Dobbiamo tradurli.

- **Ordinal Encoding:** Se c'è un ordine (es. Low < Medium < High) → 1, 2, 3.
- **One-Hot Encoding:** Se NON c'è ordine (Nominale, es. Colori, Città). Crea una colonna per ogni valore (es. `is_Red`, `is_Blue`) con 0 o 1.

```
from sklearn.preprocessing import OneHotEncoder

# handle_unknown='ignore': se nel test set appare una categoria mai vista, la ignora (tutti 0) invece di dare errore.
# sparse_output=False: restituisce un array numpy leggibile invece di una matrice sparsa compressa.
ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
```

Listing 4: OneHotEncoder

3. Feature Scaling (Mettere tutto sulla stessa scala)

Teoria: Molti algoritmi calcolano la "distanza" tra i punti (es. KMeans, KNN, SVM). Se una feature è in *metri* (1-10) e l'altra in *millimetri* (1000-10000), quella in millimetri dominerà il calcolo solo perché ha numeri più grandi.

Lo scaling porta tutto in un range comparabile (solitamente media 0 e varianza 1).

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# NOTA: Si fa fit() solo sul Train Set per evitare il "Data Leakage"!
# X_train_scaled = scaler.fit_transform(X_train)
# X_test_scaled = scaler.transform(X_test)
```

Listing 5: StandardScaler

Data Visualization & Exploration (EDA)

Questa fase serve a capire la struttura dei dati prima di applicare i modelli. Le domande d'esame tipiche richiedono di individuare: *outliers*, *imbalanced distributions* e correlazioni con il target.

1. Boxplot: Distribuzioni e Outliers

Teoria: Il Boxplot riassume la distribuzione di una variabile numerica.

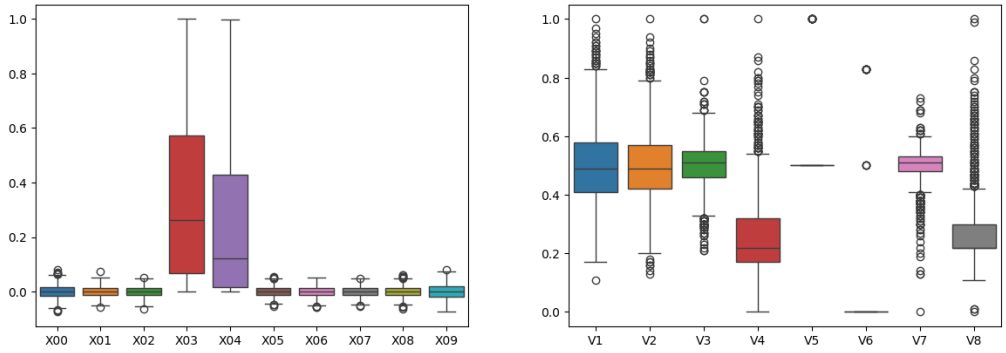
- **Box (Scatola):** Contiene il 50% centrale dei dati (dal 25° al 75° percentile).
- **Linea nel mezzo:** La Mediana (il valore centrale).
- **Baffi (Whiskers):** Si estendono fino a 1.5 volte l'IQR (intervallo interquartile).
- **Punti esterni:** Sono gli **Outliers** (valori anomali molto alti o bassi).

```
import matplotlib.pyplot as plt
import seaborn as sns

# Imposta lo stile
sns.set(style="whitegrid")
```

```
6
7  # Boxplot per visualizzare gli outliers
8  plt.figure(figsize=(12, 6))
9  # Orient='h' li fa orizzontali (piu' leggibili se hai tante features)
10 sns.boxplot(data=df, orient='h')
11 plt.title("Distribution & Outliers Analysis")
12 plt.show()
```

Listing 6: Generare Boxplot per tutte le feature



(a) Boxplot 1 (b) Boxplot 2

Figura 1: Confronto tra i due Boxplot, uno con molti outliers e uno con pochi

Come commentare all'esame (Template): *"The boxplots show that features 'Age' and 'Income' have several outliers beyond the upper whisker. The variable 'Count' shows a skewed distribution (the median is not in the center of the box)."*

2. Heatmap: Correlazioni

Teoria: Misura quanto due variabili si muovono insieme (Indice di Pearson, da -1 a 1).

- **1:** Correlazione positiva perfetta (se sale A, sale B).
- **-1:** Correlazione negativa perfetta (se sale A, scende B).
- **0:** Nessuna relazione lineare.

Approfondimento Teorico: Indice di Pearson (r)

L'indice di Pearson misura la forza e la direzione della relazione **lineare** tra due variabili continue.

Formula:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

Sostanzialmente è il rapporto tra la Covarianza di X e Y e il prodotto delle loro Deviazioni Standard.

Interpretazione dei Valori:

- $+1$ **(Correlazione Positiva Perfetta):** Tutti i punti giacciono su una retta che sale. Se X aumenta, Y aumenta in proporzione fissa.
- -1 **(Correlazione Negativa Perfetta):** Tutti i punti giacciono su una retta che scende. Se X aumenta, Y diminuisce.
- 0 **(Assenza di Correlazione Lineare):** Non c'è alcuna relazione *lineare*. Le variabili potrebbero essere indipendenti o legate in modo complesso.

Perché usiamo `abs()` nel codice? Quando cerchiamo feature predittive, ci interessa la *forza* della relazione, non la direzione. Una feature con correlazione -0.9 è preziosissima (molto predittiva), tanto quanto una con $+0.9$. Una con 0.01 è inutile (rumore).

Trappola d'Esame: Relazioni Non Lineari

Attenzione: Una correlazione di Pearson pari a 0 **non** significa che le variabili non sono collegate. Significa solo che non c'è una linea retta che le unisce.
Esempio: Se $Y = X^2$ (una parabola perfetta), l'indice di Pearson sarà ≈ 0.
Conclusione: Guardare sempre il grafico (scatterplot) prima di fidarsi ciecamente del numero!

Requisito d'Esame: Spesso si chiede di scartare le feature con correlazione assoluta col target < 0.15.

```python
# Calcola la matrice di correlazione
corr_matrix = df.corr()

# Visualizza la Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()

# Esempio: Trovare feature poco correlate con il Target (es. 'price')
target = 'price'
# abs() serve per considerare sia correlazioni positive che negative
low_corr_features = corr_matrix.index[abs(corr_matrix[target]) < 0.15].tolist()
print("Features to drop:", low_corr_features)
```

Listing 7: Heatmap e Filtro Correlazione

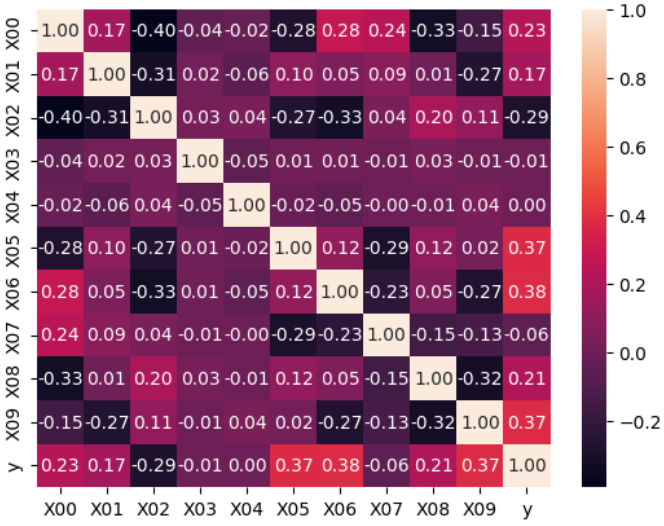


Figura 2: più il colore è chiaro, più la correlazione è alta

Come commentare all'esame: *"From the heatmap, we observe that 'Feature A' has a strong positive correlation (0.85) with the target, while 'Feature B' is weakly correlated (< 0.15) and should be dropped to reduce noise."*

Train-Test Split (Attenzione a Stratify!)

Teoria: Dividiamo i dati per simulare come il modello si comporterà nel mondo reale.

- **Regressione:** Split casuale semplice.
- **Classificazione:** Usare `stratify=y` è fondamentale se le classi sono sbilanciate (es. 90% 'No', 10% 'Sì'). Garantisce che Train e Test abbiano le stesse proporzioni.

```python
# 1. Separare Features (X) e Target (y)
target_name = 'price' # O 'class', cambia in base al dataset
X = df.drop(columns=[target_name])
y = df[target_name]

# 2. Split
# test_size=0.2 (20% test, 80% train)    lo standard
# random_state=42 rende tutto riproducibile
# stratify=y SOLO per CLASSIFICAZIONE (rimuovere per regressione)

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y # CANCELLARE QUESTA RIGA SE E' REGRESSIONE!
)

print(f"Shape Train: {X_train.shape}, Shape Test: {X_test.shape}")
```

Listing 8: Divisione Dati Corretta

Clustering: KMeans & Ottimizzazione

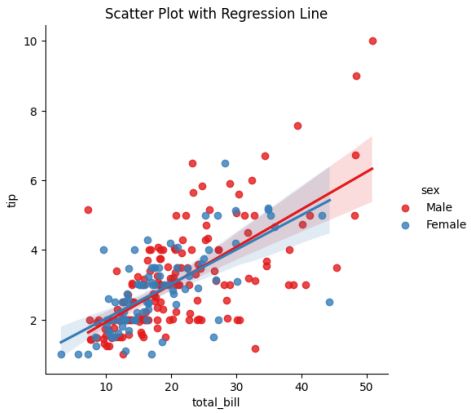
1. Concetti Chiave: Inerzia e Elbow Method

Inerzia (Inertia): È la somma delle distanze al quadrato tra ogni punto e il centro del suo cluster (centroide).

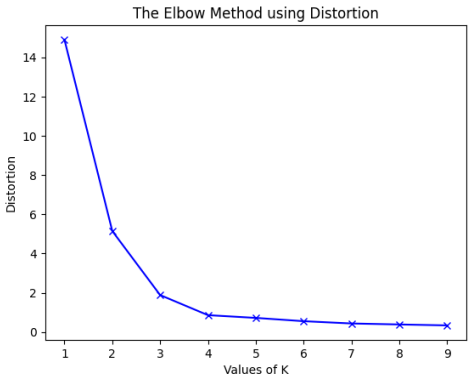
- Misura quanto i cluster sono "compatti".
- **Obiettivo:** Vogliamo un'inerzia bassa.
- **Il Problema:** Più aumentiamo K (numero di cluster), più l'inerzia scende (fino a 0 se K = numero di punti). Non possiamo solo minimizzarla, dobbiamo trovare un compromesso.

Elbow Method (Metodo del Gomito): Serve a scegliere il K ottimale. Grafichiamo l'inerzia al variare di K.

- Quando la curva scende ripida, aggiungere un cluster migliora molto il modello.
- Quando la curva si appiattisce (il "gomito"), aggiungere cluster non dà più grandi benefici ma complica solo il modello. Quel punto di flesso è il K ideale.



(a) Scatter Plot



(b) Elbow method, here, k = 3

2. Codice Completo: Elbow Method e Scatter Plot

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.preprocessing import StandardScaler


# Se non hai X, scommenta: X = df.select_dtypes(include=np.number).dropna()
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# --- PUNTO 4: K-MEANS OPTIMIZATION ---
print("--- 4. K-MEANS OPTIMIZATION ---")

# 1. Ricerca del miglior K (Constraints: k >= 3)
range_n_clusters = range(3, 11) # Da 3 a 10
best_k = -1
best_score = -1
best_model = None
best_labels = None

print(f"{'K':<5} | {'Silhouette Score'}")
print("-" * 25)

for k in range_n_clusters:
    # Hyperparameters: n_init='auto' o 10 per stabilit , random_state fisso
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=10, random_state=42)
    cluster_labels = kmeans.fit_predict(X_scaled)

    silhouette_avg = silhouette_score(X_scaled, cluster_labels)
    print(f"{k:<5} | {silhouette_avg:.4f}")

    if silhouette_avg > best_score:
        best_score = silhouette_avg
        best_k = k
        best_model = kmeans
        best_labels = cluster_labels

print("-" * 25)
print(f"Miglior schema trovato: K = {best_k} con Silhouette = {best_score:.4f}")

# 2. Mostrare gli Iperparametri (Richiesto esplicitamente)
print("\n--- Hyperparameters del modello migliore ---")
print(best_model.get_params())

# 3. Visualizzazione (Silhouette Plot + Distribuzione)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# --- PLOT A: Silhouette Plot dei cluster ---
ax1.set_xlim([-0.1, 1])
ax1.set_ylim([0, len(X_scaled) + (best_k + 1) * 10])

sample_silhouette_values = silhouette_samples(X_scaled, best_labels)
y_lower = 10

for i in range(best_k):
    ith_cluster_silhouette_values = sample_silhouette_values[best_labels == i]
    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / best_k)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower = y_upper + 10

ax1.set_title(f"Silhouette Plot per i vari cluster (K={best_k})")
ax1.set_xlabel("Coefficiente di Silhouette")
ax1.set_ylabel("Etichetta Cluster")
ax1.axvline(x=best_score, color="red", linestyle="--", label="Media")
ax1.legend()
```

```
79
80 # --- PLOT B: Distribuzione delle label (Istogramma) ---
81 unique, counts = np.unique(best_labels, return_counts=True)
82 ax2.bar(unique, counts, color=cm.nipy_spectral(unique.astype(float) / best_k), alpha=0.7)
83 ax2.set_title(f"Distribuzione elementi per Cluster (K={best_k})")
84 ax2.set_xlabel("Cluster ID")
85 ax2.set_ylabel("Numero di Record")
86 ax2.set_xticks(unique)
87
88 # Aggiunge il conteggio sopra le barre
89 for i, v in enumerate(counts):
90     ax2.text(unique[i], v + 1, str(v), ha='center')
91
92 plt.tight_layout()
93 plt.show()
```

Listing 9: KMeans: Ricerca K ottimale e Visualizzazione

Supervised Learning: Regressione, Classificazione e Tuning

In questa sezione affrontiamo l'addestramento dei modelli (con etichette, o *target*) e l'ottimizzazione degli iperparametri (GridSearch), tipici dei task finali dell'esame.

1. Il concetto di Grid Search e Cross-Validation

Teoria: Non possiamo sapere a priori quali parametri siano i migliori per un modello (es. quanto deve essere profondo un albero decisionale?).

- **GridSearch:** Prova "a forza bruta" tutte le combinazioni possibili di parametri che definiamo in una griglia.
- **Cross-Validation (CV):** Per evitare che il modello sia fortunato su un singolo split di dati, la CV divide il Train Set in K parti (fold). Allena su $K-1$ e valida sull'ultima. Ripete il processo K volte e fa la media dei punteggi.

2. Il "Mistero" del `neg_mean_squared_error`

Una domanda classica di teoria riguarda il parametro `scoring`.

Perché usiamo il valore negativo dell'errore?

- Le funzioni di ottimizzazione di `scikit-learn` (come `GridSearchCV`) sono programmate per **massimizzare** un punteggio (score). Cercano sempre il numero più alto possibile (es. Accuratezza 99% $>$ 90%).
- L'errore (MSE o RMSE), invece, deve essere **minimizzato** (un errore di 0 è perfetto).
- **Soluzione:** Usiamo il negativo.

$$\text{Errore } 10 \rightarrow \text{Score } -10$$

$$\text{Errore } 2 \rightarrow \text{Score } -2$$

Poiché -2 è matematicamente più grande (più a destra sulla retta dei numeri) di -10, l'algoritmo "massimizzando" il numero negativo sta in realtà minimizzando l'errore.

3. Codice Template: Regressione con GridSearch

Questo blocco è valido per i task che chiedono: *"optimize the depth... searching for minimum RMSE with cross-validation"*.

```
1 from sklearn.model_selection import GridSearchCV
2 from sklearn.tree import DecisionTreeRegressor
3 from sklearn.metrics import mean_squared_error, r2_score
4 import numpy as np
5
6 # 1. Definizione del modello base (fissa il random_state!)
7 dt = DecisionTreeRegressor(random_state=42)
```

```python

# 2. Definizione della griglia di parametri (Dizionario)
# Le chiavi devono corrispondere ESATTAMENTE ai parametri della funzione
param_grid = {
    'max_depth': [3, 5, 10, 20, None],      # Profondit  dell'albero
    'min_samples_split': [2, 5, 10],        # Minimo nodi per dividere
    'min_samples_leaf': [1, 2, 4]           # Minimo nodi in una foglia
}

# 3. Configurazione GridSearch
# cv=5: Divide i dati in 5 parti per testare la robustezza
# scoring='neg_mean_squared_error': Cerca di minimizzare l'errore
grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,  # Usa tutti i processori disponibili
    verbose=1   # Mostra il progresso
)

# 4. Addestramento (Solo sul TRAIN set!)
print("Inizio ricerca parametri ottimali...")
grid_search.fit(X_train, y_train)

# 5. Recupero dei risultati migliori
print("\n--- Risultati Grid Search ---")
print("Migliori parametri trovati:", grid_search.best_params_)

# best_score_    il Negativo del MSE medio in Cross-Validation.
# Dobbiamo cambiarlo di segno e farne la radice per avere l'RMSE.
best_cv_rmse = np.sqrt(-grid_search.best_score_)
print(f"Miglior RMSE in Cross-Validation: {best_cv_rmse:.4f}")

# 6. Test sul Test Set (Valutazione Finale)
# grid_search.predict usa automaticamente il miglior modello trovato
y_pred = grid_search.predict(X_test)

final_mse = mean_squared_error(y_test, y_pred)
final_rmse = np.sqrt(final_mse)
print(f"RMSE finale sul Test Set: {final_rmse:.4f}")
```

Listing 10: GridSearch per DecisionTreeRegressor

4. Feature Selection basata sulla Correlazione

Spesso viene chiesto (es. Task 4 dell'esame) di rifare il modello scartando le colonne poco correlate.

```python
# Calcolo correlazione con il target
target_col = 'price' # Sostituisci col nome vero del target
correlations = df.corr()[target_col].abs() # Valore assoluto!

# Soglia definita dall'esame (es. 0.15)
threshold = 0.15

# Seleziona le feature da tenere (quelle > 0.15)
selected_features = correlations[correlations > threshold].index.tolist()

# Rimuovi il target dalla lista delle feature X
if target_col in selected_features:
    selected_features.remove(target_col)

print(f"Feature selezionate ({len(selected_features)}): {selected_features}")

# Creazione nuovi dataset ridotti
X_train_reduced = X_train[selected_features]
X_test_reduced = X_test[selected_features]

# Ora puoi ri-addestrare il modello su questi nuovi X_train_reduced
```

Listing 11: Rimuovere colonne con bassa correlazione

5. Classificazione e Metriche (Se richiesto)

Se l'esame chiede di predire una classe (es. "Sì/No" o "Tipo A/B/C") invece di un numero, cambia la metrica di `scoring`.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Nota: Classifier invece di Regressor
clf = DecisionTreeClassifier(random_state=42)

param_grid_clf = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10]
}

# Scoring cambia! Usiamo 'accuracy' o 'f1_macro'
grid_search_clf = GridSearchCV(
    clf,
    param_grid_clf,
    cv=5,
    scoring='accuracy' # O 'f1_macro' per classi sbilanciate
)

grid_search_clf.fit(X_train, y_train)

print("Best Accuracy:", grid_search_clf.best_score_)
print("Best Params:", grid_search_clf.best_params_)
```

Listing 12: GridSearch per Classificazione

DBSCAN (Density-Based Clustering)

A differenza del KMeans, il DBSCAN non richiede di specificare il numero di cluster (K). Funziona in base alla **densità** dei punti.

Quando usarlo?

- Quando i cluster hanno forme strane (non sferiche, es. lune, anelli).
- Quando ci sono molti *outliers* (rumore) che vogliamo ignorare.

Parametri Critici:

- `eps` (Epsilon): Il raggio del cerchio intorno a ogni punto. Se è troppo piccolo, nessuno è vicino a nessuno (tutto rumore). Se è troppo grande, tutti sono collegati (un solo cluster gigante).
- `min_samples`: Quanti vicini deve avere un punto per essere considerato un "Core Point" (parte centrale di un cluster).

```python
from sklearn.cluster import DBSCAN

# Nota: DBSCAN    molto sensibile allo Scaling! Assicurati di usare X_scaled.
# eps=0.5 e min_samples=5 sono valori di default, ma vanno quasi sempre cambiati.
dbscan = DBSCAN(eps=0.5, min_samples=5)
cluster_labels = dbscan.fit_predict(X_scaled)

# DBSCAN assegna etichetta -1 al "Rumore" (punti scartati)
n_clusters = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
n_noise = list(cluster_labels).count(-1)

print(f"Cluster trovati: {n_clusters}")
print(f"Punti di rumore (Noise): {n_noise}")

# Visualizzazione (Gestendo il colore del rumore)
plt.figure(figsize=(8, 6))
unique_labels = set(cluster_labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]

for k, col in zip(unique_labels, colors):
    if k == -1:
        col = [0, 0, 0, 1] # Nero per il rumore

    class_member_mask = (cluster_labels == k)
    xy = X_scaled[class_member_mask]

    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)
```

```
29
30 plt.title('DBSCAN Clustering')
31 plt.show()
```

Listing 13: DBSCAN Clustering

Cheat Sheet: Interpretazione dei Risultati

Questa sezione serve per rispondere alle domande "Comment the results".

1. Metriche di Regressione (Numeri)

- **RMSE (Root Mean Squared Error):** È l'errore medio nella stessa unità di misura del target.
  Esempio: Se preveggo prezzi di case e RMSE = 50.000, sbaglio in media di 50k euro. **Più basso è, meglio è.**
- **R2 Score (R-Squared):** Spiega quanta varianza dei dati il modello ha catturato.
  Va da 0 a 1 (o negativo se il modello è disastroso).
  Intervalli: > 0.7 Buono, < 0.3 Scarso.

2. Metriche di Classificazione (Categorie)

- **Accuracy:** Percentuale di risposte esatte. Attenzione: se le classi sono sbilanciate (es. 99 sani, 1 malato), l'accuracy è ingannevole!
- **Confusion Matrix:**

$$\begin{bmatrix} VN & FP \\ FN & VP \end{bmatrix}$$

  Vogliamo i numeri alti sulla **diagonale principale** (Veri Negativi e Veri Positivi). I numeri fuori diagonale sono gli errori.

3. Commenti Standard per l'Esame

"The optimized Decision Tree performed better than the Linear Regression, showing a lower RMSE (12.5 vs 18.2). This suggests the relationship between features and target is non-linear."
"The DBSCAN found 3 clusters and identified 15 points as noise (outliers), which KMeans forced into the nearest cluster, reducing purity."

Modelli Avanzati: Random Forest

Teoria: Il Random Forest è un metodo "Ensemble". Invece di un solo albero (che rischia l'overfitting), ne costruisce tanti (es. 100) e fa votare ognuno. La risposta finale è la media (Regressione) o la maggioranza (Classificazione).

```
1  from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
2
3  # Esempio Classificatore
4  # n_estimators=100: Crea 100 alberi
5  rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
6
7  # Esempio Griglia per GridSearch con Random Forest
8  param_grid_rf = {
9      'n_estimators': [50, 100, 200], # Quanti alberi?
10     'max_depth': [5, 10, None],     # Quanto profondi?
11     'min_samples_split': [2, 5]
12 }
13
14 # Si usa nel GridSearch esattamente come il DecisionTree
```

Listing 14: Random Forest (Classifier o Regressor)

Modello Alternativo: K-Nearest Neighbors (KNN)

Teoria: Il KNN non "impara" una formula. Per predire un nuovo punto, guarda i K punti più vicini nel training set.

- Se è Classificazione: prende la classe di maggioranza dei vicini.
- Se è Regressione: fa la media dei valori dei vicini.

Nota: Richiede assolutamente lo **Scaling** dei dati!

```
from sklearn.neighbors import KNeighborsClassifier

# n_neighbors    il parametro 'K'
knn = KNeighborsClassifier(n_neighbors=5)

param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],  # Dispari per evitare pareggi nel voto
    'weights': ['uniform', 'distance'] # 'distance' d   pi   peso ai vicini stretti
}
```

Listing 15: KNN Classifier

Interpretazione del Modello: Feature Importance

Spesso nel punto "Comment the results" è utile dire quale variabile ha influito di più. Questo codice funziona solo con Decision Trees e Random Forest.

```
# Assumiamo che 'best_model' sia il tuo albero/foresta gi   addestrato
import pandas as pd

# Ottieni l'importanza
importances = best_model.feature_importances_
feature_names = X_train.columns # I nomi delle colonne

# Crea un DataFrame per ordinare i dati
feat_imp_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feat_imp_df = feat_imp_df.sort_values(by='Importance', ascending=False)

# Grafico a barre
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feat_imp_df, palette='viridis')
plt.title("Quali feature contano di pi ?")
plt.show()
```

Listing 16: Grafico Feature Importance

Commento d'Esame: *"From the Feature Importance plot, we see that 'Income' is the most dominant predictor (0.6 importance), while 'Age' has almost no impact on the decision. This suggests focusing business logic on Income."*