WasteGone-AI

Link al repository GitHub

January 2025

Autori:
Alessia Gatto (Matricola: 0512117767)
Giovanni Croce (Matricola: 0512116900)
Elisa Picilli (Matricola: 0512118328)

Contents

1 Introduzione: sistema attuale e sistema proposto

La gestione dei rifiuti rappresenta un tema centrale nella sostenibilità ambientale e nel miglioramento della qualità della vita urbana. Attualmente, molti sistemi di raccolta e smaltimento non riescono a garantire un'interazione personalizzata e realmente efficiente con i cittadini, limitandosi a fornire informazioni generiche o servizi poco integrati. In questo contesto, emergono soluzioni innovative che sfruttano l'intelligenza artificiale per migliorare la gestione dei rifiuti e ottimizzare l'esperienza degli cittadini.

Nonostante l'utilità di molte applicazioni per la gestione dei rifiuti, poche offrono un sistema capace di prevedere i trend locali di conferimento dei materiali riciclabili. La sfida è individuare un sistema evoluto che utilizzi tecniche di intelligenza artificiale per analizzare il numero di chili di rifiuti differenziati e non prodotti in tutti i comuni della regione Campania, consentendo una pianificazione più efficiente della raccolta e una gestione ottimizzata delle risorse.

2 Descrizione dell'agente

2.1 Obiettivi

Lo scopo del sistema è quello di realizzare un agente intelligente capace di prevedere la quantità di rifiuti differenziati e non, che sarà conferita in una specifica zona, in un anno, utilizzando dati e algoritmi di intelligenza artificiale per ottimizzare la gestione della raccolta e migliorare l'efficienza operativa.

2.2 Specifica PEAS

Definiamo ora la specifica PEAS.

Performance	Accuratezza delle previsioni sulla quantità di rifiuti differenziati e non, prodotti in una determinata zona della Campania in un anno, in modo da avere un miglioramento nella gestione della raccolta.
Environment	Dati storici e correnti su volumi di rifiuti per zone geografiche campane. • **Parzialmente osservabile**: Fattori esterni come eventi imprevisti non sono completamente noti. • **Stocastico**: Le quantità di rifiuti dipendono da variabili umane e ambientali. • **Sequenziale**: Le previsioni si basano su dati temporali. • **Dinamico**: I dati cambiano con il tempo e le abitudini. • **Continuo**: I volumi di rifiuti sono valori numerici continui. • **Multi-agente**: Coinvolge contributi di molteplici utenti e comuni.
Actuators	Dashboard e report per visualizzare previsioni e supportare decisioni.
Sensors	Sistemi che raccolgono dati su quantità di rifiuti.

Table 1: Tabella della specifica PEAS

2.3 Analisi del problema

Il problema della gestione predittiva dei rifiuti viene affrontato attraverso l'utilizzo di un algoritmo di intelligenza artificiale progettato per prevedere la quantità di rifiuti differenziati e non che sarà maggiormente conferita in una specifica zona della Campania. Questo approccio permette di ottimizzare la frequenza di raccolta, evitando viaggi inutili o sottoutilizzati, e di allocare mezzi e personale in modo più efficiente nelle aree con maggiore produzione di rifiuti. Inoltre, l'algoritmo contribuisce a prevenire l'accumulo eccessivo nei cassonetti, riducendo la necessità di interventi straordinari e favorendo una gestione più sostenibile e pianificata.

L'algoritmo analizza dati storici relativi alla gestione dei rifiuti, come il totale dei rifiuti prodotti, i chilogrammi di rifiuti differenziati e non differenziati per ciascun comune della Campania, insieme a parametri come il numero di abitanti. Sulla base di queste informazioni, il sistema è in grado di identificare i trend complessivi di produzione dei rifiuti e fornire previsioni affidabili sulla quantità totale di rifiuti differenziati e non, che sarà conferita in una specifica zona campana.

Sebbene richieda una solida infrastruttura di raccolta e analisi dei dati, questo approccio rappresenta una soluzione per supportare una pianificazione più efficiente.

3 Raccolta, analisi e preprocessing dei dati

3.1 Scelta del dataset

Per la creazione del modello di machine learning, si è deciso di utilizzare un dataset esistente che raccoglie dati sui rifiuti conferiti dai comuni della regione Campania. Questo dataset, contiene informazioni dettagliate sui rifiuti ed è stato scelto per la sua completezza e per la possibilità di estrarre conoscenze utili per il nostro obiettivo. I dati sono stati analizzati e adattati alle esigenze specifiche del progetto, garantendo la loro coerenza e applicabilità al modello predittivo.

Dopo una ricerca approfondita, è stato individuato un dataset relativo all'anno 2023 che è sembrato particolarmente adatto al progetto ma non completo in quanto si è ritenuto fosse fondamentale analizzare dati su più anni. Si è quindi deciso di selezionare 3 dataset (relativi al 2023, 2022, 2021) ed unirli in un unico dataset. Il tutto è stato selezionato dal portale ufficiale della Regione Campania dedicato agli Open Data. Questo portale è stato creato per garantire la disponibilità online dei dati regionali.

3.2 Analisi e scrematura del dataset

I dataset scelti sono reperibili ai seguenti link: dataset del 2023 , dataset del 2022 , dataset del 2021.

Il dataset sulla raccolta differenziata dei rifiuti urbani dei comuni della Regione Campania rappresenta uno strumento prezioso per analizzare la gestione dei rifiuti sul territorio regionale. Esso contiene dati dettagliati sulla quantità di raccolta differenziata, espressa in chilogrammi, e sui rifiuti urbani indifferenziati, anch'essi riportati in chilogrammi. Inoltre, il dataset fornisce i livelli percentuali di raccolta differenziata e il tasso di riciclaggio. La Regione Campania, attraverso la Direzione Generale 50 17 00, agisce sia come titolare sia come editore e autore del dataset, sottolineando l'importanza e l'affidabilità delle informazioni riportate. Il dataset completo presenta features non necessarie per lo scopo del progetto e per tale motivo sono stati effettuati dei tagli verticali. Inoltre per eliminare righe che contenevano valori nulli per lo stesso comune in uno o più anni considerati sono stati effettuati dei tagli orizzontali.

3.3 Formattazione dei dati

A questo punto è iniziato il processo di trasformazione dei dati in una forma che potesse essere più adatta al modello.

3.3.1 Aggregazione delle informazioni

A tale scopo si è deciso di aggregare i 3 dataset originali relativi agli anni 2021, 2022, 2023 in un unico dataset completo che raccoglie per ogni riga un comune e le relative informazioni riguardanti il conferimento dei rifiuti.

```python
import pandas as pd
import os
# Caricamento dei file CSV con gestione del separatore di migliaia
df_2021 = pd.read_csv('dataset/dataset2021.csv', sep=';', thousands='.')
df_2022 = pd.read_csv('dataset/dataset2022.csv', sep=';', thousands='.')
df_2023 = pd.read_csv('dataset/dataset2023.csv', sep=';', thousands='.')

# Aggiunta della colonna "Anno" per ciascun dataset
df_2021['Anno'] = 2021
df_2022['Anno'] = 2022
df_2023['Anno'] = 2023

# Concatenazione dei dataset in un unico dataframe
df_unico = pd.concat([df_2021, df_2022, df_2023], ignore_index=True)

# Salvataggio del dataset combinato in un nuovo file CSV
df_unico.to_csv('dataset/dataset_completo.csv', index=False, sep=';', float_format='%.3f')

# Ricaricamento del dataset per verificare il contenuto
dataset_completo = pd.read_csv('dataset/dataset_completo.csv', sep=';', thousands='.')
```

Il codice utilizza la libreria pandas per gestire e manipolare file CSV contenenti dati su più anni. Inizialmente, vengono caricati tre dataset relativi agli anni 2021, 2022 e 2023, specificando che il separatore di colonna è il punto e virgola (;) e che il separatore delle migliaia è il punto (.). Successivamente, a ciascun dataset viene aggiunta una colonna denominata "Anno", contenente l'anno corrispondente.

Dopo questa operazione, i tre dataset vengono concatenati in un unico DataFrame, ignorando gli indici originali per creare un dataset uniforme. Questo DataFrame combinato viene poi salvato in un nuovo file CSV con il medesimo separatore di colonna, specificando che i valori in virgola mobile devono essere formattati con tre cifre decimali. Infine, il codice ricarica il dataset appena creato per verificarne il contenuto, utilizzando le stesse impostazioni di lettura iniziali.

3.3.2 Filtraggio dei dati

Si è deciso di analizzare eventuali righe con valori nulli e features che non contribuiscono al raggiungimento dell' obiettivo.

```python
# Trova i valori nulli (NaN)
nan_mask = dataset_completo.isna()

# Trova i valori uguali a "-"
dash_mask = dataset_completo.applymap(lambda x: x == " -   " or x == " N.C. " or x == "NC")

# Somma i valori nulli (NaN) e i valori uguali a "-"
nan_count = nan_mask.sum()
dash_count = dash_mask.sum()

# Combina i risultati
total_count = nan_count + dash_count

total_count

# Colonne da verificare
columns_to_check = [
    "Kg di rifiuti differenziati (RDi)",
    "Kg di compostaggio domestico",
    "Kg di rifiuti non differenziati (RUind)",
    "Totale Kg di rifiuti prodotti (RDi+comp+RUind)",
    "Produzione R.U. pro capite annua in Kg",
    "%RD",
    "Tasso di riciclaggio"
]
```

```python
# Filtra le righe che contengono "N.C." in almeno una delle colonne specificate

filtered_rows = dataset_completo[dataset_completo[columns_to_check].applymap(lambda x: x == " N.C. " or x =="NC" or x == " NC ").any(axis=1)]

# Visualizza le righe filtrate
filtered_rows

# Elimina le righe trovate dal dataset completo
dataset_completo = dataset_completo[~dataset_completo.index.isin(filtered_rows.index)]

# Salva il dataset aggiornato
dataset_completo.to_csv('dataset/dataset_completo_pulito.csv', index=False, sep=';')

# Visualizza il dataset aggiornato
dataset_completo

# Eliminazione delle colonne specificate
dataset_completo = dataset_completo.drop(columns=[ 'Kg di compostaggio domestico', 'ATO'])

dataset_completo.rename(columns={'Totale Kg di rifiuti prodotti (RDi+comp+RUind)': 'Totale Kg di rifiuti prodotti (RDi+RUind)'}, inplace=True)

# Visualizzazione del dataset finale senza le colonne eliminate
dataset_completo
```

Il codice si occupa di elaborare un dataset chiamato dataset_completo. Inizialmente, seleziona tutte le righe in cui almeno una delle colonne specificate nella lista columns_to_check contiene uno dei seguenti valori: ” N.C. ”, ”NC” o ” NC ”. Questo viene fatto tramite una funzione che verifica per ogni valore nelle colonne selezionate se corrisponde a uno dei valori indicati. Le righe che soddisfano questa condizione vengono quindi filtrate e salvate nella variabile filtered_rows.

Dopo aver identificato le righe con questi valori problematici, il codice prosegue rimuovendole dal dataset originale. Utilizza l’indice delle righe filtrate per eliminarle, ottenendo così un nuovo dataset che non contiene più queste righe.

Il dataset pulito viene poi salvato in un file CSV chiamato dataset_completo_pulito.csv, in modo che le modifiche siano permanenti. Inoltre, il dataset aggiornato viene visualizzato, permettendo di vedere il risultato dell’operazione.

In seguito, il codice rimuove alcune colonne specifiche (in questo caso, le colonne ’Kg di compostaggio domestico’ e ’ATO’) dal dataset. Dopodiché, rinomina una colonna, cambiando il suo nome da

'Totale Kg di rifiuti prodotti (RDi+comp+RUind)' a 'Totale Kg di rifiuti prodotti (RDi+RUind)'.

Infine, il dataset finale, già modificato con le righe e le colonne rimosse, viene visualizzato.

3.3.3 Selezione dei dati

A questo punto si è deciso di raffinare il dataset eliminando le righe dei comuni che non appaiono per tutti e tre gli anni considerati.

```
# Conta le occorrenze di ciascun comune
comune_counts = dataset_completo['Comune'].value_counts()

# Filtra i comuni che appaiono almeno 3 volte
comuni_da_tenere = comune_counts[comune_counts >= 3].index

# Crea un nuovo dataset mantenendo solo i comuni desiderati
dataset_filtrato = dataset_completo[dataset_completo['Comune'].isin(comuni_da_tenere)]

# Salva il nuovo dataset su file (opzionale)
dataset_filtrato.to_csv('dataset/dataset_filtrato.csv', index=False, sep=';', encoding='utf-8')

# Mostra il dataset filtrato
dataset_filtrato
```

Questo codice esegue un filtro sui dati basandosi sulla frequenza di apparizione dei comuni all'interno del dataset.

Per prima cosa, viene calcolato il numero di occorrenze di ciascun comune, determinando quante volte ogni comune compare nel dataset. Successivamente, vengono selezionati solo i comuni che appaiono almeno tre volte, escludendo quelli con una presenza inferiore.

Il dataset viene quindi filtrato per mantenere solo i record appartenenti ai comuni selezionati, creando così una versione ridotta del dataset contenente solo i comuni con dati più ricchi e affidabili.

Infine, il dataset filtrato viene salvato in un nuovo file CSV per eventuali utilizzi successivi e viene visualizzato per verificarne il contenuto. Questo processo aiuta a concentrarsi solo sui comuni con dati più rappresentativi, evitando di analizzare quelli con informazioni limitate o frammentarie.

4 Algoritmo di regressione

4.1 Scelta dell'algoritmo di regressione

Una volta completata la formattazione dei dati si è optato per un algoritmo di regressione.

4.1.1 Algoritmo Random Forest

Sin da subito è stato scartato l'algoritmo di regressione lineare multipla per la molteplicità delle variabili indipendenti. L'algoritmo Random Forest, invece, presenta alcune particolarità:

- **Migliore gestione delle relazioni non lineari**: A differenza della regressione lineare multipla, che assume una relazione lineare tra variabili, Random Forest è un modello non lineare che riesce a catturare pattern complessi nei dati, offrendo una maggiore flessibilità e performance quando le relazioni tra le variabili non sono lineari.
- **Robustezza agli outlier e rumore**: Random Forest è meno sensibile agli outlier rispetto alla regressione lineare, che può essere influenzata da dati anomali. Le foreste di alberi aggregano molteplici decisioni, riducendo l'impatto di outlier o dati rumore.

- **Prestazioni robuste su grandi dataset**: Random Forest tende a performare bene su dataset di grandi dimensioni, mantenendo alta la qualità delle previsioni anche quando ci sono molte variabili o feature.
- **Overfitting controllato**: Random Forest è meno soggetto a overfitting rispetto al Gradient Boosting, che può adattarsi troppo ai dati di training se non viene regolato correttamente. L'uso di più alberi in Random Forest aiuta a generalizzare meglio sui dati non visti.
- **Riduzione dell'overfitting rispetto al Gradient Boosting**: Sebbene il Gradient Boosting sia un algoritmo molto potente, può essere sensibile all'overfitting, soprattutto quando i parametri non sono ottimizzati correttamente. Random Forest, pur essendo anch'esso un modello di ensemble, è generalmente meno incline a sovradattarsi rispetto a Gradient Boosting, soprattutto su dataset più complessi.

Si è deciso quindi di optare per l'algoritmo Random Forest che è sembrato il più giusto dato che ha fornito risultati precisi, grazie alla sua capacità di catturare relazioni non lineari e interazioni complesse tra le variabili, mantenendo al contempo una buona interpretabilità e robustezza nei confronti di overfitting.

Di seguito lo script che applica l'algoritmo Random Forest:

1. Il dataset "dataset_filtrato.csv" viene caricato in un DataFrame df utilizzando il separatore di colonna specificato come punto e virgola (;).
2. La funzione clean_percentage_column rimuove il simbolo di percentuale (%), sostituisce le virgole con i punti (per la notazione decimale) e converte i valori in numeri in formato float.
3. La funzione clean_numeric_column sostituisce i valori "-" con NaN, rimuove spazi e punti che vengono trattati come separatori di migliaia, e converte la colonna in valori numerici, gestendo eventuali errori nel processo tramite coerce (che assegna NaN in caso di errore).

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Carica il dataset specificando il separatore come punto e virgola
df = pd.read_csv('dataset/dataset_filtrato.csv', delimiter=';')

# Funzione per pulire le colonne con percentuali
def clean_percentage_column(col):
    return col.str.replace('%', '').str.replace(',', '.').astype(float)

# Funzione per pulire i numeri da separatori di migliaia e spazi
def clean_numeric_column(col):
    col = col.replace('-', np.nan)
    col = col.str.replace(' ', '').str.replace('.', '')
    return pd.to_numeric(col, errors='coerce')
```

1. La funzione remove_outliers calcola il primo e terzo quartile (Q1 e Q3) della colonna specificata, quindi calcola l'IQR (intervallo interquartile). Gli outlier vengono rimossi definendo un intervallo compreso tra 1.3 * IQR sotto Q1 e sopra Q3, e si selezionano solo i valori che rientrano in questo intervallo.

2. Le colonne '%RD' e 'Tasso di riciclaggio' vengono pulite rimuovendo i simboli di percentuale (%), sostituendo le virgole con i punti decimali, e convertendo i valori in numeri in formato float.

3. Le colonne 'Kg di rifiuti differenziati (RDi)' e 'Kg di rifiuti non differenziati (RUind)' vengono pulite rimuovendo i separatori di migliaia (spazi e punti) e gestendo i valori errati (come il simbolo "-", che viene sostituito con NaN), quindi vengono convertite in valori numerici.

```
# Funzione per rimuovere gli outlier utilizzando l'IQR
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.3 * IQR
    upper_bound = Q3 + 1.3 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Puliamo le colonne che contengono percentuali
df['%RD'] = clean_percentage_column(df['%RD'])
df['Tasso di riciclaggio'] = clean_percentage_column(df['Tasso di riciclaggio'])

# Puliamo le colonne target che contengono numeri con separatori di migliaia
df['Kg di rifiuti differenziati (RDi)'] = clean_numeric_column(df['Kg di rifiuti differenziati (RDi)'])
df['Kg di rifiuti non differenziati (RUind)'] = clean_numeric_column(df['Kg di rifiuti non differenziati (RUind)'])
```

1. Viene creato un set di addestramento (train_df) utilizzando i dati degli anni 2021 e 2022, mentre i dati del 2023 vengono separati come test set (test_df) per la valutazione del modello.

2. Le variabili indipendenti, che saranno utilizzate come input per il modello, sono definite nella lista features. Queste includono l'anno, il numero di abitanti, il tasso di riciclaggio, e la produzione di rifiuti per capita.

3. Le variabili target da prevedere sono definite come 'Kg di rifiuti differenziati (RDi)' e 'Kg di rifiuti non differenziati (RUind)'. Queste colonne verranno utilizzate come etichette per il training e il test.

4. Vengono separati i dati di allenamento e test per le variabili indipendenti (features) e per le variabili target, creando le matrici X_train, X_test per le features, e y_train_diff, y_train_non_diff, y_test_diff, e y_test_non_diff per le etichette target.

```
# Separiamo il training set (2021-2022) e il test set (2023)
train_df = df[df['Anno'].isin([2021, 2022])]  # Usa solo i dati del 2021 e 2022 per l'allenamento
test_df = df[df['Anno'] == 2023]  # I dati del 2023 sono utilizzati per la valutazione

# Variabili indipendenti (features)
features = ['Anno', 'Abitanti', '%RD', 'Tasso di riciclaggio', 'Produzione R.U. pro capite annua in Kg']

# Variabili target
target_diff = 'Kg di rifiuti differenziati (RDi)'
target_non_diff = 'Kg di rifiuti non differenziati (RUind)'

# Creiamo i set di training e test
X_train = train_df[features]
y_train_diff = train_df[target_diff]
y_train_non_diff = train_df[target_non_diff]

X_test = test_df[features]
y_test_diff = test_df[target_diff]
y_test_non_diff = test_df[target_non_diff]
```

1. Vengono rimosse le righe che contengono valori NaN nelle colonne dei target (rifiuti differenziati e non differenziati) sia nel training set che nel test set, creando i DataFrame train_df_clean e test_df_clean.

2. Gli outlier vengono eliminati dai target nelle colonne di interesse (rifiuti differenziati e non differenziati) utilizzando la funzione remove_outliers sia nel training set che nel test set. Questo aiuta a migliorare la qualità dei dati per l'allenamento e la valutazione del modello.

3. Dopo aver rimosso i NaN e gli outlier, vengono creati i nuovi set di training e test (i set "puliti") per le variabili indipendenti (X_train_clean, X_test_clean) e per i target (y_train_diff_clean, y_train_non_diff_clean, y_test_diff_clean, y_test_non_diff_clean).

```
# Rimuoviamo le righe con NaN nei target e nelle predizioni
train_df_clean = train_df.dropna(subset=[target_diff, target_non_diff])
test_df_clean = test_df.dropna(subset=[target_diff, target_non_diff])

# Rimuoviamo gli outlier per i target
train_df_clean = remove_outliers(train_df_clean, target_diff)
train_df_clean = remove_outliers(train_df_clean, target_non_diff)
test_df_clean = remove_outliers(test_df_clean, target_diff)
test_df_clean = remove_outliers(test_df_clean, target_non_diff)

# Creiamo i set puliti di training e test dopo aver rimosso gli outlier
X_train_clean = train_df_clean[features]
y_train_diff_clean = train_df_clean[target_diff]
y_train_non_diff_clean = train_df_clean[target_non_diff]

X_test_clean = test_df_clean[features]
y_test_diff_clean = test_df_clean[target_diff]
y_test_non_diff_clean = test_df_clean[target_non_diff]
```

1. Viene creato un modello RandomForestRegressor con 100 alberi (n_estimators=100) e una semina casuale fissa (random_state=42). Il modello viene poi allenato sui dati di training puliti relativi ai rifiuti differenziati (X_train_clean, y_train_diff_clean).

2. Una volta addestrato il modello, vengono effettuate le predizioni sui dati di test puliti relativi ai rifiuti differenziati (X_test_clean), generando le previsioni (y_pred_diff_clean) per il 2023.

3. Viene creato un altro modello RandomForestRegressor, anch'esso con 100 alberi e semina casuale fissa. Questo modello viene addestrato sui dati di training puliti relativi ai rifiuti non differenziati (X_train_clean, y_train_non_diff_clean).

4. Una volta addestrato il modello, vengono effettuate le predizioni sui dati di test puliti relativi ai rifiuti non differenziati (X_test_clean), generando le previsioni (y_pred_non_diff_clean) per il 2023.

```
# Crea e allena il modello Random Forest per i rifiuti differenziati
model_diff = RandomForestRegressor(n_estimators=100, random_state=42)
model_diff.fit(X_train_clean, y_train_diff_clean)

# Predizioni sul test set per i rifiuti differenziati (per il 2023)
y_pred_diff_clean = model_diff.predict(X_test_clean)

# Crea e allena il modello Random Forest per i rifiuti non differenziati
model_non_diff = RandomForestRegressor(n_estimators=100, random_state=42)
model_non_diff.fit(X_train_clean, y_train_non_diff_clean)

# Predizioni sul test set per i rifiuti non differenziati (per il 2023)
y_pred_non_diff_clean = model_non_diff.predict(X_test_clean)
```

1. Crea una copia del DataFrame test_df_clean per evitare il SettingWithCopyWarning
2. Aggiunge le colonne di predizione dei rifiuti differenziati e non differenziati al DataFrame test_df_clean.

```
# Creiamo una copia di test_df_clean per evitare il SettingWithCopyWarning
test_df_clean = test_df_clean.copy()

# Aggiungiamo le predizioni al DataFrame di test pulito per il 2023
test_df_clean['Predizione Rifiuti Differenziati'] = y_pred_diff_clean
test_df_clean['Predizione Rifiuti Non Differenziati'] = y_pred_non_diff_clean
```

1. Calcola e stampa l'errore quadratico medio (MSE) per i rifiuti differenziati.
2. Calcola e stampa l'errore quadratico medio (MSE) per i rifiuti non differenziati.
3. Calcola e stampa l'errore assoluto medio (MAE) per i rifiuti differenziati.
4. Calcola e stampa l'errore assoluto medio (MAE) per i rifiuti non differenziati.
5. Calcola e stampa il coefficiente di determinazione (R^2) per i rifiuti differenziati.
6. Calcola e stampa il coefficiente di determinazione (R^2) per i rifiuti non differenziati.

```python
# Calcoliamo l'errore quadratico medio per i rifiuti differenziati
mse_diff_clean = mean_squared_error(y_test_diff_clean, y_pred_diff_clean)
print(f'MSE per i rifiuti differenziati per il 2023: {mse_diff_clean}')

# Calcoliamo l'errore quadratico medio per i rifiuti non differenziat
mse_non_diff_clean = mean_squared_error(y_test_non_diff_clean, y_pred_non_diff_clean)
print(f'MSE per i rifiuti non differenziati per il 2023: {mse_non_diff_clean}')

# Calcoliamo il MAE per i rifiuti differenziati
mae_diff_clean = mean_absolute_error(y_test_diff_clean, y_pred_diff_clean)
print(f'MAE per i rifiuti differenziati per il 2023: {mae_diff_clean}')

# Calcoliamo il MAE per i rifiuti non differenziati
mae_non_diff_clean = mean_absolute_error(y_test_non_diff_clean, y_pred_non_diff_clean)
print(f'MAE per i rifiuti non differenziati per il 2023: {mae_non_diff_clean}')

# Calcoliamo il R² per i rifiuti differenziati
r2_diff_clean = r2_score(y_test_diff_clean, y_pred_diff_clean)
print(f'R² per i rifiuti differenziati per il 2023: {r2_diff_clean}')

# Calcoliamo il R² per i rifiuti non differenziati
r2_non_diff_clean = r2_score(y_test_non_diff_clean, y_pred_non_diff_clean)
print(f'R² per i rifiuti non differenziati per il 2023: {r2_non_diff_clean}')
```

1. Crea un DataFrame per il 2024 con le stesse caratteristiche di test_df, utilizzando i dati della popolazione e altri parametri.
2. Utilizza il modello per prevedere la quantità di rifiuti differenziati per il 2024.
3. Utilizza il modello per prevedere la quantità di rifiuti non differenziati per il 2024.

```python
# Crea un DataFrame per i dati del 2024 con le stesse caratteristiche
df_2024 = pd.DataFrame({
    'Anno': [2024] * len(test_df),  # L'anno per cui vogliamo fare le previsioni
    'Abitanti': test_df['Abitanti'],  # Utilizzando gli stessi abitanti del 2023 (esempio)
    '%RD': test_df['%RD'],
    'Tasso di riciclaggio': test_df['Tasso di riciclaggio'],
    'Produzione R.U. pro capite annua in Kg': test_df['Produzione R.U. pro capite annua in Kg'],
})

# Predizione per i rifiuti differenziati per il 2024
pred_diff_2024 = model_diff.predict(df_2024)

# Predizione per i rifiuti non differenziati per il 2024
pred_non_diff_2024 = model_non_diff.predict(df_2024)
```

1. Aggiunge la colonna 'Comune' da test_df al DataFrame df_2024.
2. Inserisce le predizioni dei rifiuti differenziati e non differenziati nel DataFrame df_2024.
3. Calcola e aggiunge la colonna "Totale kg rifiuti prodotti" sommando le due predizioni.
4. Salva il DataFrame df_2024 in un file CSV chiamato predizioni_rifiuti_2024.csv.

```python
# Aggiungiamo la colonna 'Comune' dal test_df al DataFrame df_2024
df_2024['Comune'] = test_df['Comune']

# Aggiungiamo le predizioni per il 2024
df_2024['Predizione Rifiuti Differenziati'] = pred_diff_2024
df_2024['Predizione Rifiuti Non Differenziati'] = pred_non_diff_2024

# Aggiungi la colonna "totale kg rifiuti prodotti" come somma delle due colonne
df_2024['Totale kg rifiuti prodotti'] = df_2024['Predizione Rifiuti Differenziati'] + df_2024['Predizione Rifiuti Non Differenziati']

# Salviamo il risultato in un file CSV
df_2024.to_csv('predizioni_rifiuti_2024.csv', index=False)

print("Predizioni per i rifiuti nel 2024 salvate nel file 'predizioni_rifiuti_2024.csv'")
```

1. Crea due grafici di regressione che confrontano i valori reali e predetti per i rifiuti differenziati e non differenziati.
2. Utilizza punti per rappresentare i dati e una linea rossa per indicare la perfetta corrispondenza tra predizioni e valori reali.

```python
# Grafico di regressione per i rifiuti differenziati
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_test_diff_clean, y_pred_diff_clean, color='blue')
plt.plot([y_test_diff_clean.min(), y_test_diff_clean.max()], [y_test_diff_clean.min(), y_test_diff_clean.max()], color='red', lw=2)
plt.title('Rifiuti Differenziati - Predizione vs Realità')
plt.xlabel('Valori reali')
plt.ylabel('Valori predetti')

# Grafico di regressione per i rifiuti non differenziati
plt.subplot(1, 2, 2)
plt.scatter(y_test_non_diff_clean, y_pred_non_diff_clean, color='green')
plt.plot([y_test_non_diff_clean.min(), y_test_non_diff_clean.max()], [y_test_non_diff_clean.min(), y_test_non_diff_clean.max()], color='red', lw=2)
plt.title('Rifiuti Non Differenziati - Predizione vs Realità')
plt.xlabel('Valori reali')
plt.ylabel('Valori predetti')

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

5 Conclusione

5.1 Metriche di valutazione

Il risultato del calcolo delle metriche è il seguente:

- **MAE (Mean Absolute Error)**: In media, le previsioni del modello differiscono di circa 77167.45 kg rispetto ai valori reali per i rifiuti differenziati mentre per i rifiuti non differenziati circa 55211.62kg.
- **RMSE (Root Mean Squared Error)**: L'errore quadratico tra le previsioni e i valori reali è 135774.46 kg per i rifiuti differenziati e 110375.75 kg per i rifiuti non differenziati.
- **R^2 (Coefficiente di determinazione)**: Il modello rappresentare circa il 98% della variabilità dei dati per i rifiuti differenziati, mentre per i rifiuti non differenziati circa il 91.8%, il che indica una buona accuratezza.

```
MSE per i rifiuti differenziati per il 2023: 18434704407.810738
MSE per i rifiuti non differenziati per il 2023: 12182807195.610119
MAE per i rifiuti differenziati per il 2023: 77167.45016997168
MAE per i rifiuti non differenziati per il 2023: 55211.62781869688
R² per i rifiuti differenziati per il 2023: 0.9807345689640219
R² per i rifiuti non differenziati per il 2023: 0.9183057880740408
Predizioni per i rifiuti nel 2024 salvate nel file 'predizioni_rifiuti_2024.csv'
```

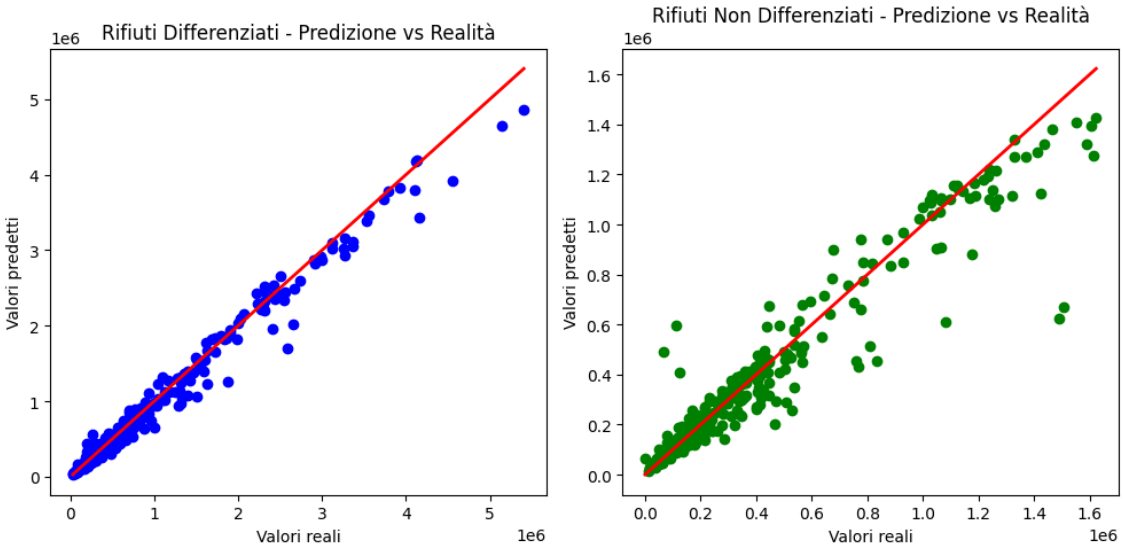
5.2 Grafico di regressione

Un grafico di regressione è uno strumento visivo che aiuta a valutare le prestazioni di un modello di regressione confrontando i valori reali (test set) con i valori predetti (dal modello).

Le componenti del grafico di regressione sono:

- **Asse X**: Rappresenta i valori reali della variabile target (quelli osservati nel test set).
- **Asse Y**: Rappresenta i valori predetti dal modello.
- **Punti nel grafico**: Ogni punto nel grafico rappresenta una coppia (valore reale, valore predetto).
- **Linea rossa tratteggiata**: È la linea ideale di riferimento, dove ogni valore reale è uguale a quello predetto.

Se un punto si trova sulla linea, significa che il valore predetto è esattamente corretto. Se un punto si trova sopra la linea, significa che il modello ha sovrastimato il valore reale. Se un punto si trova sotto la linea, significa che il modello ha sottostimato il valore reale.



5.3 Considerazioni finali

Un R^2 alto suggerisce che la regressione è molto efficace nel prevedere il totale dei rifiuti (un valore vicino a 1 indica un modello ben adattato). Il valore del MAE è relativamente basso rispetto alla scala dei valore, quindi le previsioni sono abbastanza precise. Il valore di RMSE potrebbe indicare la presenza di alcuni errori molto grandi in alcune previsioni, che potrebbero essere influenzati da outliers.

6 Glossario

- **Algoritmo di Intelligenza Artificiale**: Un insieme di regole e procedure che permettono a un sistema di analizzare dati, apprendere dai pattern e prendere decisioni autonome. Gli algoritmi di IA possono essere supervisionati (come le reti neurali) o non supervisionati (come il clustering).
- **Machine Learning**: Un ramo dell'intelligenza artificiale che permette ai computer di apprendere dai dati senza essere esplicitamente programmati. I modelli di machine learning identificano schemi nei dati e migliorano le loro prestazioni nel tempo attraverso l'esperienza.
- **Modello**: Un insieme di regole, equazioni o reti neurali utilizzato per apprendere dai dati e fare previsioni.
- **Specifica PEAS**: Un modello utilizzato per definire un agente intelligente basato su quattro componenti:
  - **P (Performance)**: criteri per valutare le prestazioni dell'agente.
  - **E (Environment)**: l'ambiente in cui l'agente opera.
  - **A (Actuators)**: le azioni che l'agente può compiere.
  - **S (Sensors)**: i sensori che l'agente utilizza per raccogliere informazioni.
- **Feature**: "caratteristica", un attributo o una variabile indipendente nel dataset che contribuisce alla previsione della variabile target.
- **Dataset**: Un insieme di dati organizzati in forma strutturata (tabella, CSV, database) utilizzati per addestrare e testare modelli di machine learning.
- **Regressione**: Una tecnica statistica e di machine learning usata per modellare la relazione tra variabili. Stima come una variabile dipendente cambia in base a una o più variabili indipendenti.
- **Random Forest**: Un algoritmo di apprendimento automatico basato su un insieme di alberi decisionali. Ogni albero viene costruito su un campione casuale dei dati di addestramento, e per ogni nodo viene selezionato un sottoinsieme casuale delle caratteristiche.
- **Variabili Dipendenti**: Le variabili che il modello cerca di prevedere. Nella regressione, la variabile dipendente è quella influenzata dalle variabili indipendenti.
- **Variabili Indipendenti**: Le variabili usate per spiegare o prevedere il valore della variabile dipendente. Nella regressione lineare, sono i fattori che influenzano l'output del modello.
- **Outlier**: Un punto dati che si discosta significativamente dalla distribuzione generale dei dati. Nella regressione, gli outlier possono influenzare negativamente il modello, alterando la pendenza della retta di regressione e riducendo l'accuratezza delle previsioni.
- **Metriche di Valutazione**: Strumenti utilizzati per misurare le prestazioni di un modello di machine learning.
- **Z-score**: Misura statistica che indica di quante deviazioni standard un valore si discosta dalla media della distribuzione dei dati. È utile per individuare outlier. Si calcola come:

$$Z = \frac{x - \mu}{\sigma}$$

  dove x è il valore osservato, μ la media e σ la deviazione standard.
- **Training Set**: La porzione del dataset usata per addestrare il modello. Contiene esempi da cui l'algoritmo impara i pattern e le relazioni tra le variabili.

- **Test Set**: Un sottoinsieme del dataset utilizzato per valutare le prestazioni del modello dopo l'addestramento. Il modello non ha mai visto questi dati prima, quindi forniscono un'indicazione dell'accuratezza delle previsioni.
- **DataFrame**: Una struttura dati di pandas che organizza le informazioni in formato tabellare con righe e colonne, simile a un foglio Excel. È molto usata per manipolare e analizzare dati in Python.