STUDIO DELLA QUALITÀ DELL'ARIA NEL 2023

COMPONENTI DEL GRUPPO:

Vanessa Barbaro [MAT.725617]: v.barbaro3@studenti.uniba.it

Stefano Todaro [MAT.720335]: s.todaro4@studenti.uniba.it

A.A 2023/2024

Github: https://github.com/VanessaBarb/ProgettoIcon/tree/main

Introduzione

L'obbiettivo della tesi è studiare e comprendere come i maggiori inquinanti presenti nell'aria influiscano e in che peso sulla qualità dell'aria, ed inoltre analizzare come (o se) questa giochi un ruolo in alcuni fattori climatici

Come quantificatore per il calcolo della qualità dell'aria è stato selezionato l'**Air Quality Index** (**AQI**), ovvero un sistema per tradurre le misurazioni delle concentrazioni di inquinanti, a volte confuse o poco intuitive, in una scala di facile comprensione per rappresentare chiaramente il rischio per la salute posto dall'inquinamento dell'aria ambiente. L'AQI traduce le misurazioni delle concentrazioni di questi inquinanti in una scala che va da 0 a 500. I valori dell'AQI possono ricadere in 6 categorie distinte:

- Buono
- Moderato
- Malsano
- Molto malsano
- Pericoloso

Il calcolo dell'AQI si basa su 6 principali sostanze inquinanti presenti nell'aria:

- PM2.5
- PM10
- NO_2
- SO_2
- O_3
- CO

Il dataset selezionato (e di conseguenza lo studio) analizza le misurazioni degli inquinanti e i dati climatici (temperatura, umidità, velocità del vento) in 20 grandi città nel mondo nell'arco di tutto il 2023.

Requisiti funzionali:

Il progetto è stato realizzato in Python3.12 in quanto offre un vasto numero di librerie che permettono di trattare e manipolare i dati in modo semplice. Il file **Unsupervised.py** deve essere compilato indipendentemente. Stessa cosa per il file **Ontology.csv**, ma unicamente dopo aver estratto il file **Global_Aqi_Ontology.xml** dall'omonimo file zip.

IDE utilizzato: PyScripter

Librerie utilizzate:

- Pandas
- Sklearn
- Matplotlib
- Seaborn
- Imblearn
- Numpy
- Pgmpy

Creazione del Dataset

Il dataset, già di base, conteneva già la maggior parte delle informazioni utili. I dati presenti erano informazioni geografiche e temporali accompagnate dai relativi valori ambientali e climatici;

- **City**: Riporta le città in cui sono state effettuate le misurazioni. Questa feature contiene le 20 più grandi e principali città nel mondo, come ad esempio Parigi, Tokyo, New York, ecc.
- **Country**: Contiene i paesi di appartenenza delle città. Ogni città appartiene ad un paese differente.
- **Date**: Indica le date in cui sono state effettuate le misurazioni. Queste sono state effettuate lungo tutto l'arco del 2023.
- **PM2.5**: Riporta le misurazioni medie nell'arco della giornata di **particolato fine**. Il particolato fine (Particulate Matter PM) è costituito da particelle solide e liquide aventi diametro aerodinamico variabile fra 0,1 e circa 100 µm che tendono a rimanere sospese in aria. Il termine PM2.5 è relativo alle particelle con diametro aerodinamico inferiore o uguale ai 2.5 µm . Generalmente tali particelle sono costituite da una miscela di elementi quali carbonio, fibre, metalli. nitrati, solfati, composti organici , materiale inerte e particelle liquide.
- **PM10**: La definizione di questo elemento è molto simile alla precedente, con la differenza che questa feature misura la quantità media giornaliera di particolato fine delle particelle di diametro aerodinamico inferiore o uguale ai 10 µm (1 µm = 1 millesimo di millimetro)
- **NO_2**: Mostra la misurazione giornaliera media del **biossido di azoto**, si forma in massima parte in atmosfera per ossidazione del monossido (NO), inquinante principale che si forma nei processi di combustione. Le emissioni da fonti antropiche derivano sia da processi di combustione che da processi produttivi senza combustione (produzione di acido nitrico, fertilizzanti azotati, ecc.)
- **SO2**: Riporta la quantità media di **biossido di zolfo**. Questo si orma nel processo di combustione per ossidazione dello zolfo presente nei combustibili solidi e liquidi. Le fonti di emissione principali sono legate alla produzione di energia, agli impianti termici, ai processi industriali e al traffico. L'SO_2 è il principale responsabile delle "piogge acide".
- **CO**: Misura la quantità media giornaliera di **monossido di carbonio**, un gas inodore e incolore che si forma dalla combustione incompleta degli idrocarburi presenti in carburanti e combustibili. Le concentrazioni in aria di questo inquinante possono essere ben correlate all'intensità del traffico.
- **O3**: Misura la quantità di **ozono** mediamente presente nell'aria. L'ozono è un gas incolore ed inodore, la sua presenza al livello del suolo dipende fortemente dalle condizioni meteoclimatiche. Le concentrazioni di Ozono più elevate si registrano normalmente nelle zone distanti dai centri abitati ove minore è la presenza di sostanze inquinanti con le quali, a causa del suo elevato potere ossidante, può reagire.
- **Temperature**: Riporta la temperatura media calcolata nella giornata in gradi Celsius (C°)
- **Humidity**: Indica il livello di umidità medio giornaliero misurato in percentuale

- **Wind Speed**: Mostra la velocità media del vento misurata in chilometri orari (km/h)

Dati gli obbiettivi dello studio è stato necessario aggiungere due colonne fondamentali, ovvero la colonna rappresentante l'AQI, aggiunta con il nome di **Air_Quality**, e la colonna che contenente la categoria dell'indice precedente, nominata **Air_Quality_Label**.

Per il calcolo dell'AQI è stato selezionato il metodo standard:

per prima cosa si calcolare il valore di AQI per ciascun inquinante, tramite la formula standard

$$AQI = (Chi - Clo)(Ihi - Ilo) \times (C - Clo) + Ilo$$

Dove:

- **C**: è la concentrazione misurata dell'inquinante.
- $\mathbf{C_{lo}}$: è il limite inferiore dell'intervallo di concentrazione in cui rientra la concentrazione **C**.
- $\mathbf{C_{hi}}$: è il limite superiore dell'intervallo di concentrazione in cui rientra la concentrazione **C**.
- $\mathbf{I_{lo}}$: è il valore di AQI corrispondente a $\mathbf{C_{lo}}$
- $\mathbf{I_{hi}}$: è il valore di AQI corrispondente a $\mathbf{C_{hi}}$

I limiti sono specifici per sostanza.

Una volta fatte queste misurazioni si prende il valore di AQI maggiore fra tutti gli inquinanti e questo determinerà la qualità dell'aria.

Una volta effettuati i calcoli per ogni riga presente nel dataset è stato necessario classificare ogni valore di AQI riportato, e per fare ciò si sfrutta la tabella di classificazione, che per ogni categoria descrivente la qualità dell'aria determina un range. Dipendente in quale range ricade il l'indice di qualità dell'aria è possibile determinare a quale categoria inscrivere la misurazione.

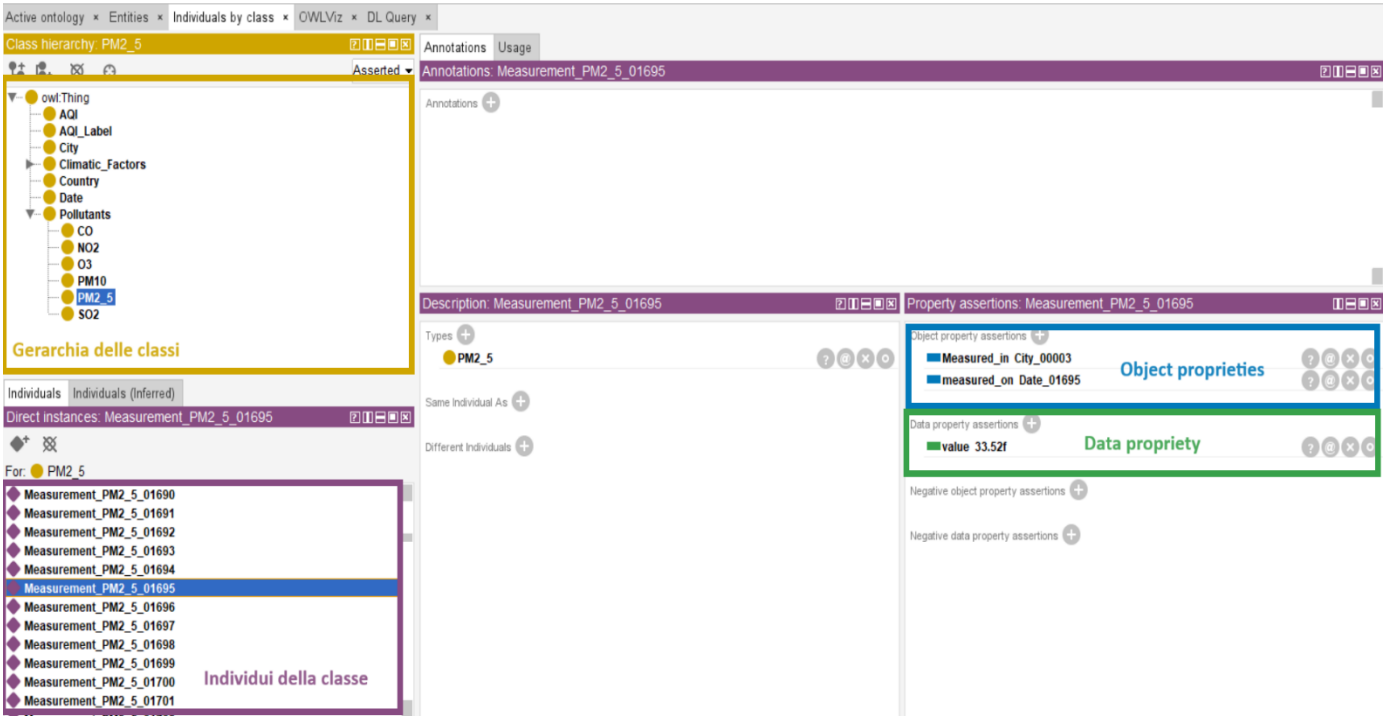
Valori dell'indice di qualità dell'aria(AQI)	Livelli di preoccupazione per la salute	
0-50	Sconosciuto	Misurazioni scarse, non affidabili
51-100	Buona	La qualità dell'aria è considerata soddisfacente e l'inquinamento atmosferico presenta rischi minimi o nulli
101-150	Moderato	La qualità dell'aria è accettabile; tuttavia, per alcuni inquinanti può esserci un problema di salute moderato per un numero molto limitato di persone che sono insolitamente sensibili all'inquinamento atmosferico
151-200	Malsano	I residenti appartenenti a gruppi sensibili(per patologie) possono avere effetti sulla salute. È improbabile che gli effetti siano seri per tutta la popolazione
201-300	Molto malsano	Ognuno può iniziare a sperimentare effetti sulla salute; i membri di gruppi sensibili possono avere effetti sulla salute più gravi
>300	Pericoloso	Allerta salute: chiunque può essere soggetto a effetti più gravi sulla propria salute

Tramite queste modifiche è stato creato il nuovo dataset **"globalAirNew.csv"** utilizzato per addestrare il modello ed effettuare le misurazioni.

Ontologia

In informatica, con il termine **ontologia** si intende una struttura formale che descrive in modo esplicito e organizzato un dominio di interesse, definendo le entità, i concetti e le relazioni che lo caratterizzano. Queste sono caratterizzate da una struttura fortemente gerarchica, con una relazione "is a" che va a collegare concetti più specifici con quelli generali. Queste sono rappresentati da linguaggi formali come ad esempio OWL. Queste forniscono un'ottima base di conoscenza.

Per la creazione dell'ontologia è stato utilizzato il tool **Protegè-5.6.3**. Si tratta di un editor open-source specializzato nella creazione sistemi intelligenti.



L'ontologia è stata organizzata in classi, ognuna indicante le caratteristiche delle misurazioni (come luogo, data, inquinanti, caratteristiche climatiche ecc.). Ognuno di queste classi conterrà degli individui caratterizzati da delle **data proprieties**, che ne indicano le informazioni principali, e saranno in relazione tra loro, ove necessario, da delle **object proprieties**.

Per popolare l'ontologia con tutti i dati utili provenienti dal dataset è stata creata una funzione apposita: **populate_ontology()**.

```
#vengono inseriti individui e le loro relazioni in base al dataset
def populate_ontology(df):
    df = df.sort_values(by=['Date'])
    #dizionari contenenti il nome delle città, dei paesi e delle label
    city_dict = {}
    country_dict = {}
    label_dict = {}
    #indice per numerare le misurazioni
    i=0
    #indici per numerare città,paesi e label
    cr=0
    ci=0
    l=0
    for index, row in df.iterrows():
        i+=1
        padded_index = str(i).zfill(5)
        measurement_pm25_uri = URIRef(NS[f'Measurement_PM2_5_{padded_index}'])
        g.add((measurement_pm25_uri, RDF.type, NS['PM2_5']))
        g.add((measurement_pm25_uri, NS['value'], Literal(row['PM2.5'], datatype=XSD.float)))
```

Questa funzione viene eseguita per ogni informazione all'interno del dataset. Per valori ripetuti spesso, come città, paese e categoria di AQI. Questi sono stati gestiti tramite dei dizionari appositi che mantengono le informazioni specifiche per classe, in modo da scongiurarne le ripetizioni.

```
 # Gestione dei paesi
country = row['Country']
if country not in country_dict:
    cr += 1
    padded_index_cr = str(cr).zfill(5)
    country_uri = URIRef(NS[f'Country_{padded_index_cr}'])
    country_dict[country] = country_uri
    g.add((country_uri, RDF.type, NS['Country']))
    g.add((country_uri, NS['name'], Literal(country, datatype=XSD.string)))
else:
    country_uri = country_dict[country]
```

Infine, terminato l'inserimento delle informazioni utili, si passa all'inserimento delle relazioni tra gli individui delle differenti classi

```
    g.add((measurement_pm25_uri, NS['Measured_in'], city_uri))
    g.add((measurement_pm10_uri, NS['Measured_in'], city_uri))
    g.add((measurement_no2_uri, NS['Measured_in'], city_uri))
    g.add((measurement_so2_uri, NS['Measured_in'], city_uri))
    g.add((measurement_co_uri, NS['Measured_in'], city_uri))
    g.add((measurement_o3_uri, NS['Measured_in'], city_uri))
    g.add((measurement_pm25_uri, NS['Measured_in'], city_uri))
    g.add((temperature_uri, NS['Measured_in'], city_uri))
    g.add((wind_speed_uri, NS['Measured_in'], city_uri))
    g.add((humidity_uri, NS['Measured_in'], city_uri))
    g.add((aqi_uri, NS['Measured_in'], city_uri))


    g.add((measurement_pm25_uri, NS['measured_on'], date_uri))
    g.add((measurement_pm10_uri, NS['measured_on'], date_uri))
    g.add((measurement_no2_uri, NS['measured_on'], date_uri))
    g.add((measurement_so2_uri, NS['measured_on'], date_uri))
    g.add((measurement_co_uri, NS['measured_on'], date_uri))
    g.add((measurement_o3_uri, NS['measured_on'], date_uri))
    g.add((measurement_pm25_uri, NS['measured_on'], date_uri))
    g.add((temperature_uri, NS['measured_on'], date_uri))
    g.add((wind_speed_uri, NS['measured_on'], date_uri))
    g.add((humidity_uri, NS['measured_on'], date_uri))
    g.add((aqi_uri, NS['measured_on'], date_uri))

    g.add((aqi_uri, NS['corresponds_to'], label_uri))
```

Apprendimento non supervisionato

L'apprendimento non supervisionato (unsupervised learning) utilizza gli algoritmi di machine learning per analizzare e raggruppare in cluster i dataset senza etichette. I modelli di unsupervised learning vengono utilizzati per tre attività principali, il clustering, associazione e riduzione della dimensionalità.

Clustering

È una tecnica che raggruppa i dati non etichettati in base alle similitudini e le differenze delle loro caratteristiche. Gli algoritmi di clustering possono essere raggruppati in varie tipologie, come algoritmi esclusivi, sovrapposti, gerarchici e probabilistici. Quelli più comunemente utilizzati sono gli algoritmi di tipo esclusivo e sovrapposto, noti anche come **hard** e **soft**.

Un algoritmo si definisce hard, se le classificazioni degli elementi del clustering questa avviene in maniera netta, inscrivendo ogni elemento ad un unico cluster. Mentre, un algoritmo è di tipo soft consenti ai punti di appartenere a più cluster con gradi di appartenenza differenti.

Regole di associazione

Si tratta di metodi basati su regole utilizzati principalmente per trovare le relazioni tra le variabili di un dataset. Questi metodi vengono usati principalmente nelle analisi di mercato. I metodi principali sono algoritmi utilizzati per generare regole di associazione, come Apriori, Eclat e FP-Growth.

Riduzione della dimensionalità

La riduzione della dimensionalità è una tecnica utilizzata quando il numero di caratteristiche, o dimensioni, in un determinato dataset è troppo elevato. Riduce il numero di input di dati a una dimensione gestibile preservando il più possibile l'integrità del dataset. È comunemente usata nella fase di preelaborazione dei dati e ci sono alcuni diversi metodi di riduzione della dimensionalità come l'analisi del componente principale (Principal component analysis o PCA), la decomposizione ai valori singolari (Singular value decomposition, SVD) e l'autoencoder.

Preprocessing dei dati

In un primo momento, dato l'obbiettivo dello studio, era stato deciso di effettuare una clusterizzazione delle misurazioni basate sulle temperature in relazione al valore dell'AQI. Ma questo è apparso molto poco pratico, in quanto tramite un'analisi della matrice di correlazione è apparso come, stando ai dati ritrovati, i fattori climatici siano influenzati in minima parte dalla presenza degli inquinanti nell'aria, riportando valori che rasentano lo zero.

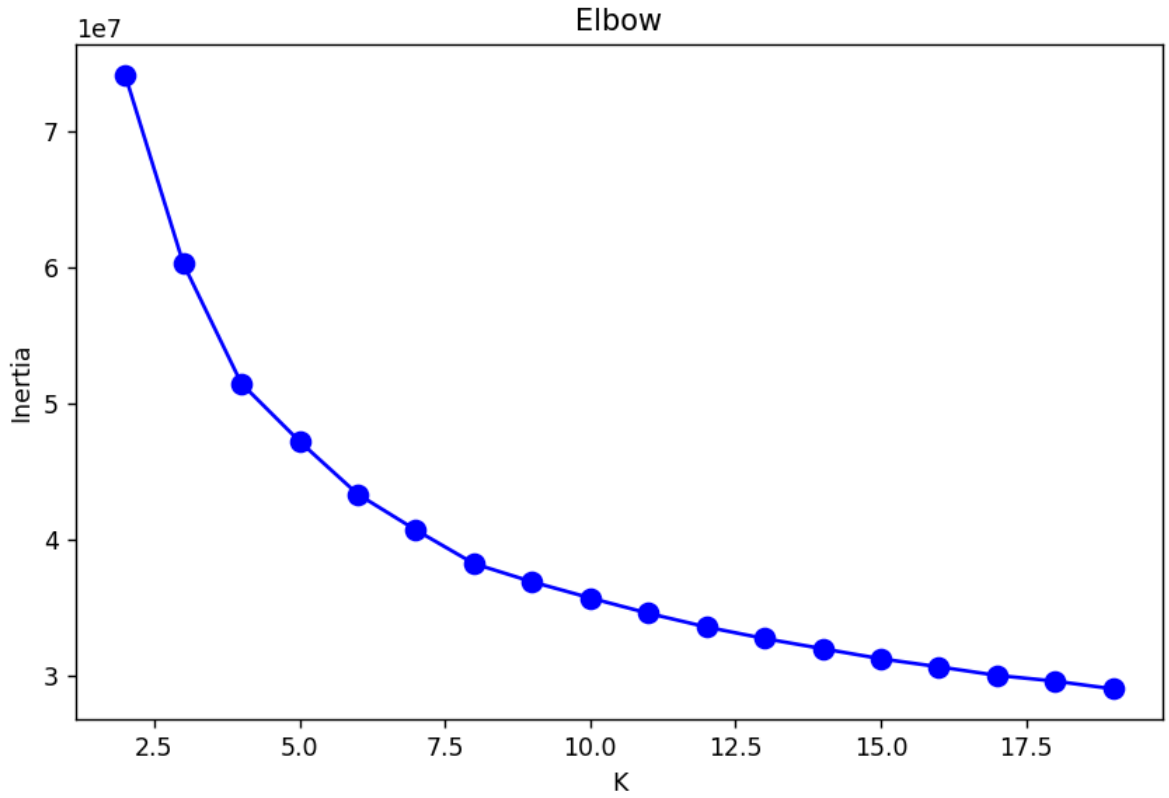
A questo punto si è deciso di clusterizzare i punti in base alle misurazioni degli inquinanti principali e la qualità dell'aria. Sempre tramite un'analisi della matrice di correlazione è stato osservato come l'elemento inquinante con maggiore impatto sulla qualità dell'aria sia il PM2.5, il quale è stato dunque scelto per la clusterizzazione dei dati.

È stato deciso di utilizzare un algoritmo di hard clustering: K-means. Il K-means è un algoritmo il cui obbiettivo è quello di clusterizzare i punti del dataset in un numero *k* specificato di cluster, calcolando per ogni punto la sua distanza dai centroidi di classe, rappresentanti i punti medi dei cluster, per poi assegnarlo al cluster con la distanza minore. Uno dei punti focali di questo algoritmo è la definizione del numero di cluster *k*, i quale è determinate per la qualità della distribuzione dei dati.

Per effettuare questi calcoli è stato necessario fare riferimento ai valori numerici del dataset.

Per il ritrovamento del valore *k*, è stato utilizzato il metodo **find_k()**, che sfrutta due algoritmi: il metodo più comunemente utilizzato, ovvero il **metodo del gomito**, in combinazione con il **silhouette score** calcolato in relazione al valore di k. Il metodo del gomito si basa sull'osservazione della diminuzione dell'inerzia all'aumentare del numero di cluster. Ciò accade in quanto, in generale, un numero maggiore di cluster rappresentano meglio la divisione dei dati. Con **inerzia** si definisce la somma delle distanze quadratiche tra ciascun punto e il centroide del cluster di appartenenza. Viene definito metodo del "gomito" in quanto nell'osservazione grafico riportante il rapporto tra inerzia e il valore di k, si seleziona il valore di k in cui il valore di inerzia rallenta la

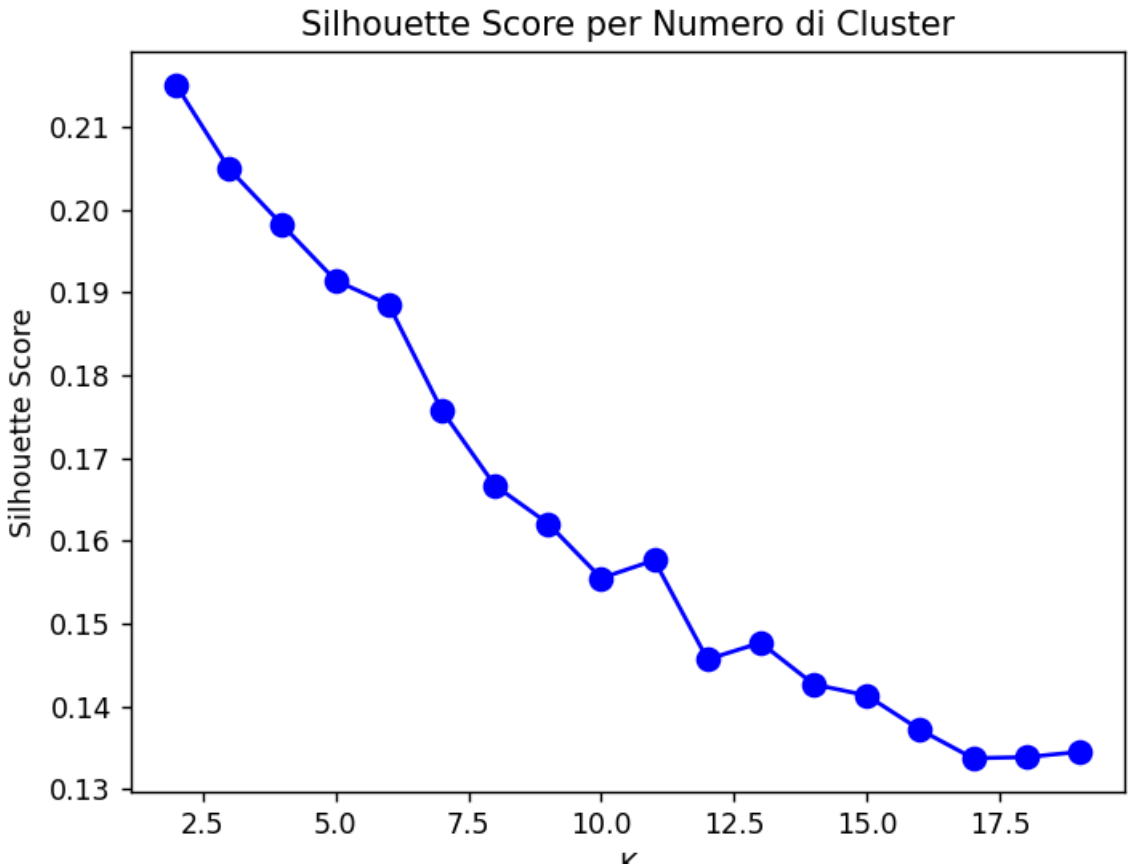
sua discesa andando a formare un certo angolo con i precedenti valori in discesa.



Osservando il grafico si nota come questo sia rappresentato quasi da una curva, rendendo ardua l'identificazione del valore *k*. Per questo è stato ritenuto necessario utilizzare il *silhouette score*.

Il *silhouette score* definisce la distanza media tra i punti assegnati ai vari cluster, ovvero definisce quanto i cluster siano ben distinguibili gli uni dagli altri. Questo viene definito tramite un coefficiente di silhouette range [-1,1], con un valore 1 indica una separazione perfetta dei cluster, un valore 0 che indica come la separazione dei cluster non sia netta e anzi ci possono essere anche delle sovrapposizioni ed infine un valore negativo può indicare perfino delle assegnazioni errate. Duque, vengono effettuate delle misurazioni calcolando l'andamento del silhouette score in

base al numero di *k* di cluster.



Si può notare come all'aumentare del numero di cluster si ha un repentino peggioramento del coefficiente di silhouette.

Mettendo i due grafici a confronto si è giunti alla conclusione che il valore ottimale per l'algoritmo è un numero di cluster *k*=3 (valore che verrà riconfermato anche da delle prove empiriche).

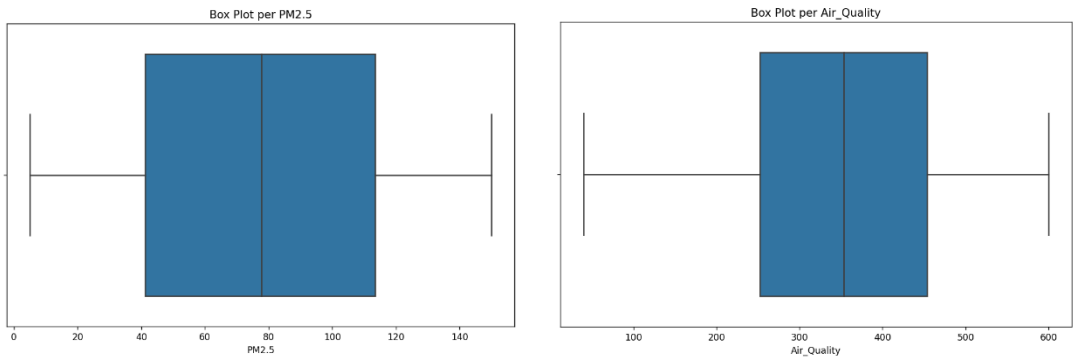
Prima di passare alla fase effettiva di clusterizzazione degli elementi del dataset, è stato necessario processare i dati in modo da ottenere dei risultati più performanti. Questo lavoro di ottimizzazione è stato seguito tramite il metodo **manage_outliers()**.

```python
#metodo per il ritrovamento e l'eliminazione degli outliers
def manage_outliers(df):
    df_no_outliers=df.copy()
    #selezione delle features di interesse
    features=df[['PM2.5','Air_Quality']]
    #si calcolano i quartili per entrambe le features e si effettua un filtraggio
    for feature in features:
        plt.figure(figsize=(10, 6))
        #visualizzazione del range di valori tramite boxplot
        sns.boxplot(x=df[feature])
        plt.title(f'Box Plot per {feature}')
        plt.show()

        #calcolo dell'IQR per la feature corrente
        Q1 = df[feature].quantile(0.25)
        Q3 = df[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # filtraggio dei dati per rimuovere gli outliers
        df_no_outliers = df_no_outliers[(df_no_outliers[feature] ≥ lower_bound) &
                                        (df_no_outliers[feature] ≤ upper_bound)]
    #si ritorna il dataset filtrato
    return df_no_outliers
```

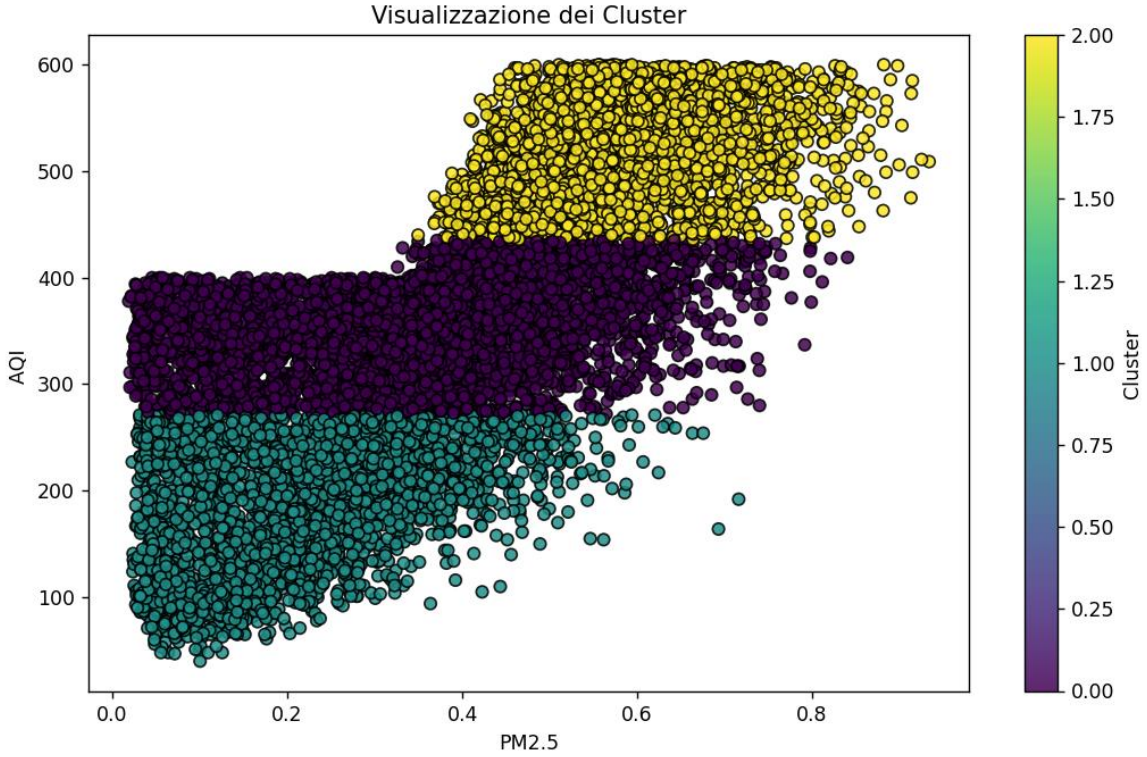
L'algoritmo di K-means è molto sensibile agli outliers, ovvero quei dati che si discostando dalla maggior parte dei valori presenti nel dataset. Per questo è stato utilizzato un metodo specifico per la gestione di questi valori. Per prima cosa sono stati creati dei grafici tramite la libreria Seaborn per visualizzazione della distribuzione dei valori principali per le classi di interesse.



In seguito, per ogni feature, sono stati calcolati i quartili, definendo il limite inferiore e superiore dei valori accettabili; i valori esterni a tale range vengono identificati come outliers e vengono rimossi dal DataFrame.

Il filtraggio dei valori viene seguito da un loro ulteriore raffinamento tramite lo scaler. La **scalarizzazione** (o **normalizzazione**) dei valori è una tecnica per trasformare le caratteristiche del dataset in un intervallo specifico e per l'ottenimento di una distribuzione più uniforme. Questo processo permette di assicurare la comparabilità dei valori e rende più accurata la misurazione delle distanze nel machine learning. Come algoritmo per la normalizzazione dei dati è stato selezionato l'algoritmo di **Unic Vector Transformation**, in quanto è il metodo è indipendente dalla magnitudine delle caratteristiche, fondando la distanza dei punti unicamente sulla direzione del vettore.

Effettuato questo ultimo processing dei dati, si passa all'esecuzione dell'algoritmo di *K-means*, ottenendo questa distribuzione:



Per avere un'idea più chiara delle performance del modello, è stato calcolato il coefficiente di silhouette score anche per cluster, ottenendo un valore di 0.59, che per quanto non sia ottimale si possa ritenere sufficientemente buono.

APPRENDIMENTO SUPERIVSIONATO

L'apprendimento supervisionato è un ramo di Machine Learning in cui il modello viene addestrato su un dataset etichettato. L'obiettivo è far sì che il modello impari a mappare gli input agli output corretti, così da poter fare predizioni accurate su nuovi dati non ancora etichettati.

Gli algoritmi di apprendimento supervisionato vengono utilizzati per problemi di classificazione (dove l'output è una categoria) e di regressione(dove l'output è un valore numerico)

L'obiettivo di questo progetto è quello di costruire un modello di classificazione per predire la qualità dell'aria basandosi su diverse misurazioni di inquinanti atmosferici. Dalla fase di feature study è emerso che gli inquinanti che contribuiscono maggiormente alla qualità dell'aria sono 'PM2.5' e 'PM10'. Utilizzeremo questi due inquinanti per le sperimentazioni sul dataset.

Sono stati implementati i seguenti modelli:

- Classificatore: Random Forest Classifier
- Predittori: Decision Tree e Random Forest

Classificatore

Un classificatore è un algoritmo di machine learning utilizzato per assegnare una categoria o classe a un insieme di dati di input. In un problema di classificazione, il modello impara a distinguere tra diverse classi basandosi sui dati di addestramento.

Il random Forest Classifier è un algoritmo di apprendimento supervisionato che utilizza una combinazione di alberi decisionali (chiamati Decision Trees) per fare delle previsioni, ognuno dei quali fa una predizione per una classe.

La classe finale assegnata a un dato è quella che riceve la maggioranza dei voti da parte degli alberi.

L'algoritmo è molto apprezzato per la sua capacità di ridurre l'overfitting, migliorando così la generalizzazione delle previsioni su dati non visti.

Predittore

Il machine learning predittivo è una branca dell'Intelligenza Artificiale che utilizza algoritmi e tecniche statistiche per prevedere risultati futuri basati su dati storici e attuali.

Questo tipo di apprendimento automatico è fondamentale per lo sviluppo di modelli predittivi, ovvero strumenti in grado di prevedere eventi futuri con una certa precisione, basandosi su pattern e tendenze presenti nei dati.

Il Random Forest può essere utilizzato per fare predizioni. Come già detto in precedenza, l'algoritmo comprende più alberi decisionali che operano in parallelo. Ciascun albero decisionale nel modello riceve un sottoinsieme casuale delle feature del dataset e un sottoinsieme casuale dei dati di addestramento.

Questo processo di creazione di alberi indipendenti e diversi permette di ottenere predizioni più accurate e robuste, poiché gli errori di un singolo albero tendono ad essere compensati dagli altri.

Il decision Tree è una struttura ad albero utilizzata come modello predittivo per algoritmi di machine learning. In un albero decisionale, ciascun nodo rappresenta una condizione su una singola feature, ogni ramo rappresenta il risultato di quella condizione, e ogni foglia rappresenta una classe finale (il risultato).

METODOLOGIA

Nel seguito si analizzano i modelli implementati e si fornisce un'analisi delle prestazioni dei modelli.

Classificatore

L'implementazione del classificatore random-forest prevede le seguenti fasi:

- Caricamento e preprocessing dei dati
  In questa fase sono stati caricati i dati del dataset con *load_data* e successivamente sono state aggiunte delle colonne al dataset per il calcolo dell'Air_Quality (AQI) e la categoria relativa all'AQI, Air_Quality_Category

```
def prepare_data(df):
    #Calcola l'aqi di tutte le righe del dataset
    df['Air_Quality'] = df.apply(calculate_overall_aqi, axis= 1)
    df['Air_Quality_Category']= df['Air_Quality'].apply(aqi_to_category)
    df.to_csv("globalAirNew.csv", index=False)
    return df
```

  Con la funzione *prepare_data* vengono create le colonne sopra citate.
- Preparazione del modello del modello

```
def prepare_model_data(df):
    #seleziona le feature e l'etichetta di target
    X= df[['PM2.5', 'PM10', 'NO2','SO2','CO', 'O3']]
    y = df['Air_Quality_Category']
    #Dividi i dati in training e l'etichetta target
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2, random_state=
    return X_train, X_test, y_train, y_test
```

  La funzione *prepare_model_data* seleziona le feature e il target. In questo caso le feature selezionate per il modello sono PM2.5, PM10, NO2, SO2, CO, O3. Il target è rappresentato dalla colonna Air_Quality_Category
  Il dataset è stato poi diviso in Training Set e Test set, con una proporzione di 80% training e 20% test
- Addestramento del modello

```python
def train_model(X_train, X_test, y_train, y_test):
    #Normalizza le feature
    scaler = StandardScaler();
    X_train = pd.DataFrame(scaler.fit_transform(X_train),columns=X_train.columns)
    #X_test = scaler.transform(X_test)

    # Applica SMOTE al dataset di training
    smote = SMOTE(k_neighbors= 2, random_state=42)
    X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

    #Rimuovi valori NaN
    X_train = pd.DataFrame(X_train).dropna().values
    X_test = pd.DataFrame(X_test).dropna().values
    y_train = y_train.dropna()
    y_test = y_test.dropna()

    #creazione e addestramento modello classificatore
    model = RandomForestClassifier(n_estimators= 100, random_state=42, class_weight= 'balanced')
    model.fit(X_train_res, y_train_res)
    print("\nDistribuzione delle classi dopo SMOTE:")
    print(y_train_res.value_counts())


    return model, scaler, X_train_res, y_train_res
```

Con la funzione *train_model* si effettua l'addestramento del modello, includendo la tecnica di Normalizzazione e la tecnica SMOTE.

Nella fase di Feature Study è stato evidenziato uno sbilanciamento del dataset, in cui la maggior parte dei dati è classificato come 'Dangerous'. Date le scarse prestazioni del modello sul dataset non bilanciato, abbiamo deciso di utilizzare il metodo SMOTE per bilanciare le classi.

Le feature sono state normalizzate utilizzando uno StandardScaler, e poi segue l'applicazione dello SMOTE

Il modello di Random Forest è stato creato e addestrato con 100 alberi.

- Valutazione del modello

  In questa fase vengono valutate le prestazioni del modello.

```python
def evaluate_model(model,df, X_test, y_test,scaler, X_train_res, y_train_res):
    X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro', zero_division=0)
    recall = recall_score(y_test, y_pred, average='macro', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='macro', zero_division=0)

    print(f"\nAccuratezza: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1-score: {f1:.2f}")

    conf_matrix = confusion_matrix(y_test, y_pred)
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predetto')
    plt.ylabel('Reale')
    plt.show()



    occurrences = df['Air_Quality_Category'].value_counts()
    # Visualizza i risultati
    print("\nOccorrenze di Air_Quality_Category: ", occurrences)

    # Controlla la distribuzione delle categorie nell'insieme di test
    print("\nDistribuzione delle categorie nel test set:")
    print(y_test.value_counts())
```

Mediante la funzione *evaluate_model*, il modello è stato valutato su un set di test, calcolando diverse metriche, tra cui accuracy, precision, recall e F1-score
È stata realizzata la matrice di confusione per consentire la visualizzazione dei risultati della classificazione.
Infine, si procede con la verifica della presenza di overfitting del modello.

```python
#verifica overfitting
def check_of(model, X_train, y_train, X_test, y_test, X_train_res, y_train_res):
    scaler= StandardScaler()
    X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
    X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)

    y_train_pred= model.predict(X_train_res)
    y_test_pred= model.predict(X_test)


    #Calcolo delle metriche sul training set
    train_accuracy = accuracy_score(y_train_res, y_train_pred)
    train_precision = precision_score(y_train_res, y_train_pred, average='weighted', zero_division=0)
    train_recall = recall_score(y_train_res, y_train_pred, average='weighted', zero_division=0)
    train_f1 = f1_score(y_train_res, y_train_pred, average='weighted', zero_division=0)


    # Calcolo delle metriche sul test set
    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted', zero_division=0)
    test_recall = recall_score(y_test, y_test_pred, average='weighted', zero_division=0)
    test_f1 = f1_score(y_test, y_test_pred, average='weighted', zero_division=0)

    # Stampa dei risultati
    print("\n\nMetriche del training set:")
    print(f"Accuratezza: {train_accuracy:.2f}")
    print(f"Precision: {train_precision:.2f}")
    print(f"Recall: {train_recall:.2f}")
    print(f"F1-score: {train_f1:.2f}")

    print("\n\nMetriche del test set:")
    print(f"Accuratezza: {test_accuracy:.2f}")
    print(f"Precision: {test_precision:.2f}")
    print(f"Recall: {test_recall:.2f}")
    print(f"F1-score: {test_f1:.2f}")
```

```python
# Eseguire la cross-validation sul training set
cv_scores = cross_val_score(model, X_train_res, y_train_res, cv=5, scoring='accuracy')

# Stampa dei risultati
print(f"\nAccuratezza Cross-validation: {cv_scores}")
print(f"\nMedia accuratezza Cross-validation: {cv_scores.mean():.2f}")
```

Di seguito si analizzano i risultati ottenuti dalla valutazione del modello classificatore Random Forest.

Per le metriche accuracy, precision, recall e F1-Score per la valutazione del modello Random Forest Classifier:

```
Accuratezza: 0.99
Precision: 0.98
Recall: 0.89
F1-score: 0.92
```

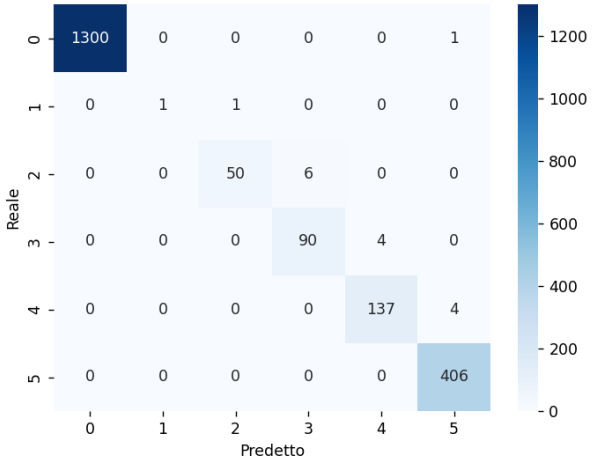
L’accuracy rappresenta la percentuale di campioni classificati correttamente dal modello rispetto al totale. L’accuratezza del 99% indica che il modello classifica correttamente il 99% dei campioni

La precisione misura la capacità del modello di evitare falsi positivi, ovvero quante delle previsioni sono effettivamente corrette. Una precisione del 98% indica che il modello ha valutato quasi precisamente i campioni.

Il recall misura la capacità del modello di identificare correttamente tutti i campioni positivi. Un recall dell'89% indica che il modello riesce ad indentificare l'89% dei campioni positivi reali, perdendone un 11%.

L'F1-score è una metrica che misura la media armonica di precision e recall. La valutazione di 0.92 indica che il modello mantiene un buon equilibrio tra le due metriche, evitando falsi positivi e la perdita eccessiva di campioni positivi.

La matrice di confusione è la seguente:



Questa matrice mostra la distribuzione delle predizioni del modello. Ogni riga rappresenta la classe reale, mentre ogni colonna rappresenta la classe predetta dal modello.

La corrispondenza è la seguente:

- Classe 0: Dangerous
- Classe 1: Good
- Classe 2: Moderate
- Classe 3: Poor
- Classe 4: Unhealty
- Classe 5: Very Unhealty

Si può notare che la classe Dangerous è la più predetta, mentre la classe Good è la meno predetta. Questo è causato da una scarsa presenza di campioni valutata come 'Good' all'interno del dataset.

Per la fase di verifica della presenza di overfitting sono state calcolate le metriche di accuratezza, precision, recall e f1-score sul training set e sul test set. Infine, è stato calcolato il cross-validation e fornita la sua media.

```
Metriche del training set:
Accuratezza: 1.00
Precision: 1.00
Recall: 1.00
F1-score: 1.00

Metriche del test set:
Accuratezza: 0.99
Precision: 0.99
Recall: 0.99
F1-score: 0.99
Accuratezza Cross-validation: [0.99561617 0.998214   0.99821371 0.99788892 0.99691458]
Media accuratezza Cross-validation: 1.00
```

Le metriche calcolate sul training set sono perfette. Ciò vuol dire che il modello si adatta perfettamente ai dati di addestramento.

Le metriche sul test set sono molto alte e molto vicine a quelle del training set. Questo suggerisce che il modello generalizza bene su dati non visti.

La cross-validation mostra prestazioni molto buone del modello, con una media di accuratezza molto alta. Questo conferma che il modello ha buone prestazioni e non è eccessivamente adattato a un particolare sottoinsieme di dati.

Dati questi risultati, si può constatare che il modello non mostra segni di overfitting ed è in grado di generalizzare bene su nuovi dati.

PREDITTORE

Per valutazione di entrambi i modelli Decision Tree e Random Forest, viene utilizzato il modulo *"ValutazioneModello.py"* per effettuare la valutazione dei modelli.

```python
#Valutazione del modello
def evaluate_model(rf,nome_modello,X_train,X_test,y_train, y_test, y_pred):
    print(f"Report di classificazione {nome_modello}:\n", classification_report(y_test, y_pred, zero_division=1))
    print(f"Accuratezza {nome_modello}: ", accuracy_score(y_test,y_pred))

    #Grafico delle distribuzioni delle predizioni
    print("Distribuzione delle predizioni nel test set: ", Counter(y_pred))
    pred_counter= Counter(y_pred)
    labels= list(pred_counter.keys())
    counts= list(pred_counter.values())

    plt.figure(figsize=(10,6))
    plt.bar(labels, counts, color='skyblue')
    plt.xlabel('Classi predette')
    plt.ylabel('Conteggio')
    plt.title("Distribuzione delle predizioni nel test set")
    plt.show()

    #Matrice di confusione
    cm= confusion_matrix(y_test, y_pred)
    disp= ConfusionMatrixDisplay(confusion_matrix= cm, display_labels= rf.classes_)
    disp.plot(cmap='Blues')
    disp.figure_.set_size_inches(12, 8)
    plt.show()

    #Cross validation
    scores= cross_val_score(rf, X_train, y_train, cv=5, scoring='accuracy')
    print("Accuratezza media della cross validation: ", scores.mean())

    print("Previsioni delle etichette Air_Quality_Category")
    print(y_pred)
```

```
comparison= pd.DataFrame({'Vero':y_test, 'Previsto': y_pred})
print("Confronto tra etichette vere e previste:")
print(comparison)

#Filtra le predizioni errate
errors= comparison[comparison['Vero'] ≠ comparison['Previsto']]
print("Campioni con errore di previsione:")
print(errors)
errors_num= (y_test ≠ y_pred).sum()
print("Numero errori: ", errors_num)

n_file= 'Confronto_previsioni_' + nome_modello +  '.csv'
comparison.to_csv(n_file,index=False)
print(f"Previsioni salvate in {n_file}")
```

Questo metodo, *evaluate_model* fornisce le seguenti informazioni:

- Report di classificazione: riepilogo delle metriche di performance del modello
- Accuratezza del modello sui dati di test
- Distribuzione delle predizioni: visualizza la distribuzione delle predizioni del modello con un grafico a barre
- Matrice di confusione, per analizzare visivamente le performance del modello
- Cross-validation, utilizzato sui dati di addestramento
- Confronto delle etichette vere e quelle previste

Nei vari file indicati nel codice, vengono memorizzate le previsioni del modello di riferimento.

Decision-Tree

Questo modello utilizza il dataset “Air_Quality_Category_Balanced.csv” che contiene il modello bilanciato, realizzato dal modulo “Smote.py” che utilizza la tecnica SMOTE per bilanciare il dataset originale.

L’implementazione del Decision-Tree prevede le seguenti fasi:

- Preparazione dei dati
  Mediante la funzione *prepare_data* vengono selezionate le feature (PM2.5 e PM10) e la variabile target (Air_Quality_Category) e si effettua la divisione del dataset in set di addestramento e set di test. Il 70% dei dati è riservato all’addestramento, mentre il 30% è utilizzato per il test.

```
def prepare_data(df):
    features= ['PM2.5', 'PM10']
    X= df[features]
    y= df['Air_Quality_Category']

    #Divisione in dati di test e dati di train
    X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.3, random_state=42)

    return X_train, X_test, y_train, y_test
```

- Addestramento modello
  La funzione *train_model* si occupa dell’addestramento del modello Decision Tree e della previsione sui dati di test:

```python
def train_model(X_train, y_train, X_test):
    DecTree= DecisionTreeClassifier(random_state=42)
    DecTree.fit(X_train, y_train)
    y_pred= DecTree.predict(X_test)

    return DecTree,y_pred
```

- Valutazione del modello.

Di seguito si analizzano i risultati ottenuti dalla fase di valutazione.

Il report di classificazione ha generato questa tabella:

```
Report di classificazione Decision-Tree:
                precision    recall  f1-score   support

     Dangerous       1.00      1.00      1.00      1935
          Good       0.98      0.99      0.98      1908
      Moderate       0.97      0.96      0.97      1874
          Poor       0.98      0.98      0.98      1921
     Unhealthy       1.00      1.00      1.00      1955
Very Unhealthy       1.00      1.00      1.00      1987

      accuracy                           0.99     11580
     macro avg       0.99      0.99      0.99     11580
  weighted avg       0.99      0.99      0.99     11580
```

Da questo report si evince che il modello ha predetto con precisione le classi “Dangerous, Unhealty e Very Unhealthy”, evitando falsi positivi.

Anche per le altre classi “Good, Moderate e Poor” la precisione è alta, con valori di 0.97-0.98

Il recall è elevato, con valori che oscillano da 0.96 a 1.00, indicando che il modello ha identificato correttamente quasi tutti i campioni delle classi.

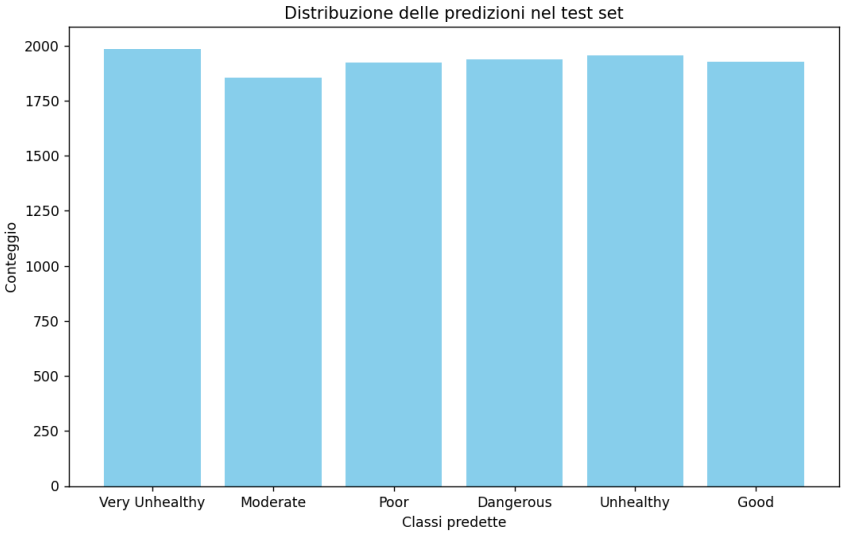
Anche l’F1-score è vicino al massimo per tutte le classi, con valori che oscillano tra 0.97 e 1.00, suggerendo un buon bilanciamento tra le metriche precision-recall.

Per quanto riguarda l’accuratezza del modello Decision-Tree:

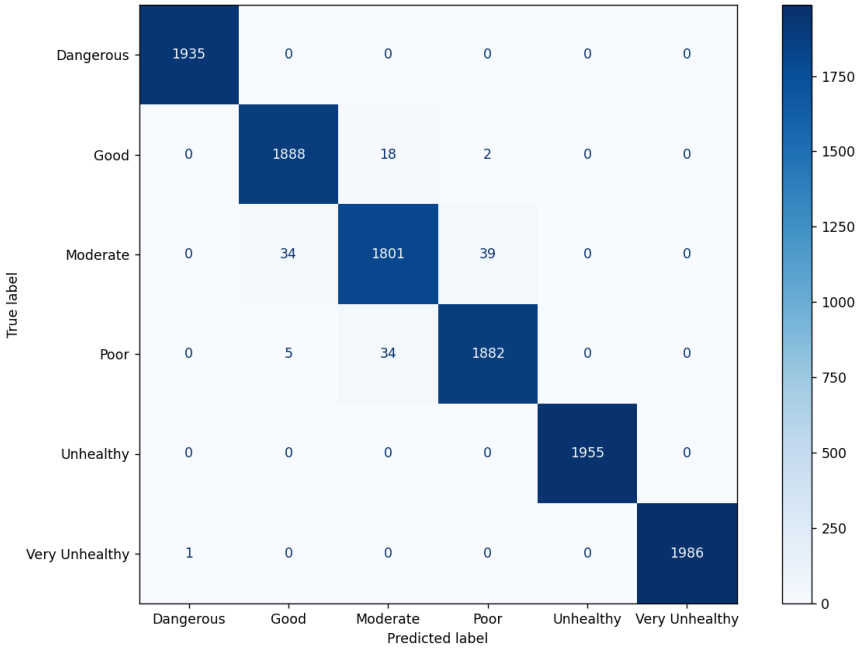
```
Accuratezza Decision-Tree:  0.9885146804835924
Distribuzione delle predizioni nel test set:  Counter({'Very Unhealthy': 1986, 'Unhealthy': 1955, 'Dangerous': 1936, 'Good': 1927, 'Poor': 1923, 'Moderate': 1853})
```

La distribuzione delle predizioni nel test set è abbastanza uniforme tra le varie classi, mettendo in evidenza che il modello non è sbilanciato verso una particolare classe.

Di seguito una rappresentazione grafica della distribuzione delle predizioni:



La matrice di confusione mostra il confronto tra le etichette previste dal modello con le etichette reali del dataset, che riassume e conferma i risultati appena esaminati.




```
Accuratezza media della cross validation:  0.985639225295375
```

L'accuratezza media della cross validation conferma la capacità del modello di generalizzare bene su dati non visti.

Nel dataset "*Confronto_previsioni_Decision-Tree.csv*" vengono mostrati anche i 133 errori di predizioni commessi dal modello Decision-Tree.

Random Forest

Per l'implementazione del modello predittore Random-Forest sono previsti i seguenti passaggi:

- Caricamento e filtraggio dei dati.

```python
def load_data():
    df=pd.read_csv(r"globalAirNew.csv")
    return df
```

La funzione *load_data* si occupa di caricare i dati dal file .csv chiamato "*globalAirNew.csv*"

```python
def filter_data(df):
    #Conta il numero di dati etichettati come "Dangerous"
    dangerous_df= df[df['Air_Quality_Category']=='Dangerous']
    dangerous_count= dangerous_df.shape[0]
    print(f"Numero di dati etichettati come 'Dangerous': {dangerous_count}")

    num_to_remove= dangerous_count//2

    #Rimuove la metà dei dati Dangerous
    dangerous_to_remove= dangerous_df.iloc[:num_to_remove]
    df_filtered = df[~df.index.isin(dangerous_to_remove.index)]

    dangerous_count_after_removal= df_filtered[df_filtered['Air_Quality_Category'] == 'Dangerous'].shape[0]
    print(f"Numero di dati etichettati come 'Dangerous' dopo la rimozione: {dangerous_count_after_removal}")

    return df_filtered
```

La funzione *filter_data* applica un filtro specifico per ridurre lo sbilanciamento delle classi nel dataset. Nello specifico conta il numero delle istanze etichettate come "Dangerous" e ne rimuove la metà. Dopo la rimozione, viene stampato il numero delle istanze rimanenti nella classe "Dangerous" per confermare il risultato del filtro.

- Preparazione dei dati

```python
def prepare_data(df_filt):
    #Divisione del dataset in feature e target
    X= df_filt[['PM2.5','PM10']]
    y= df_filt['Air_Quality_Category']

    #Divisione del dataset in training e testing
    X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2, random_state=42)
    return X_train, X_test, y_train, y_test
```

La funzione *prepare_data* divide il dataset filtrato in feature(PM2.5, PM10) e target(Air_Quality_Category). Il dataset viene poi suddiviso in dati di training(80%) e dati di test(20%).

- Addestramento del modello

```python
def train_model(X_train,X_test, y_train):
    #Pesi manuali:
    class_weights= {
        'Dangerous':1,
        'Good':100,
        'Moderate':1,
        'Poor':1,
        'Unhealthy':1,
        'Very Unhealthy':1
    }

    rf= RandomForestClassifier(random_state=42, class_weight=class_weights)
    rf.fit(X_train, y_train)
    y_pred= rf.predict(X_test)
    return y_pred, rf
```

Il metodo *train_model* crea e addestra un modello di Random Forest bilanciato.
Viene indicato un dizionario 'class_weights' che assegna un peso manuale più elevato alla classe 'Good' per cercare di bilanciare la classe meno frequente.

- Valutazione del modello.

Di seguito si analizzano i risultati derivanti dalla valutazione del modello Random-Forest

Per il report di classificazione è stata realizzata questa tabella:

```
Report di classificazione Random Forest:
                precision    recall  f1-score   support

     Dangerous       1.00      1.00      1.00       637
          Good       1.00      0.00      0.00         2
      Moderate       0.90      0.90      0.90        51
          Poor       0.95      0.96      0.96       114
     Unhealthy       1.00      1.00      1.00       147
Very Unhealthy       1.00      1.00      1.00       406

      accuracy                           0.99      1357
     macro avg       0.98      0.81      0.81      1357
  weighted avg       0.99      0.99      0.99      1357

Accuratezza Random Forest:  0.9918938835666913
```

L'accuratezza del 0.99 indica che il modello è altamente efficace nel classificare la qualità dell'aria sulla base delle caratteristiche fornite.

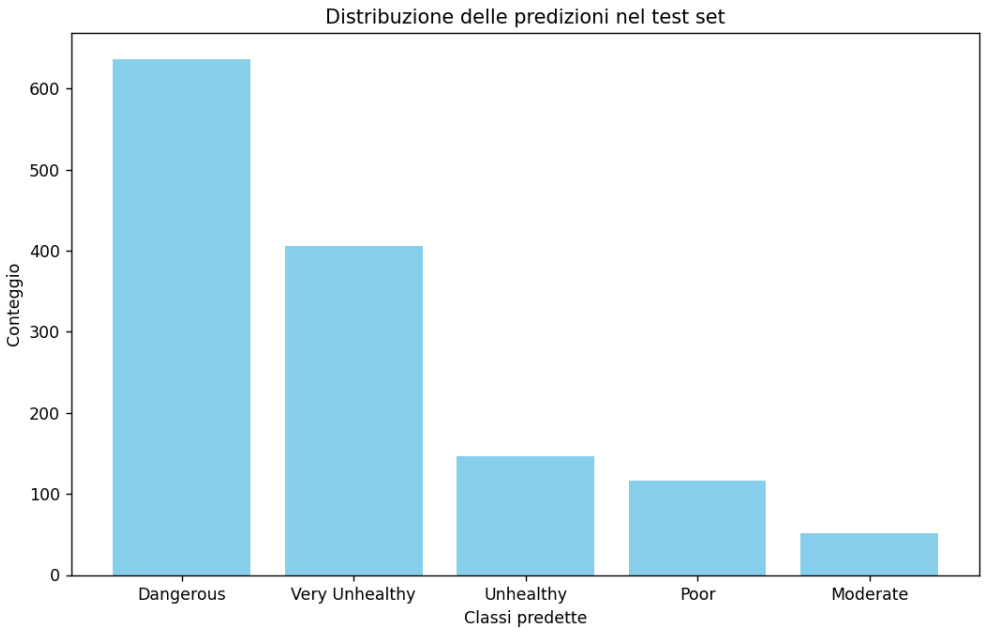
La precisione è molto alta per tutte le classi. Questo significa che, quando il modello prevede una di queste classi "Dangerous, Unhealty e Very Unhealthy", la predizione è sempre corretta.

Il recall è generalmente alto, tranne che per la classe "Good", dove è 0. Questo indica che il modello, nonostante il bilanciamento, non è in grado di identificare correttamente le istanze di "Good" presenti nel set di test.

Di conseguenza l'F1-score è elevate per quasi tutte le classi, tranne che per "Good" a cui associa il valore "0".

```
Accuratezza Random Forest:  0.9918938835666913
Distribuzione delle predizioni nel test set:  Counter({'Dangerous': 637, 'Very Unhealthy': 406, 'Unhealthy': 147, 'Poor': 116, 'Moderate': 51})
```

Nonostante il bilanciamento del test set e i tentativi di porre diversi pesi alla classe “Good”, questa non viene gestita a causa delle pochissime istanze nel dataset.
Di seguito la rappresentazione visiva della distribuzione delle predizioni nel test set:



La matrice di correlazione mostra visivamente le prestazioni analizzate poco fa.