Pollutants and Climate: Predictive Modeling in NYC

Victor Gavrilenko Ilay Cohen Shay Harush
Lidor Mashiach
Department of Software and Information System
BGU
Student IDs: [209406255, 206515744 , 314804287 , 209280098]

June 2025

Research Question

How do daily variations in pollutant concentrations relate to changes in temperature in New York City before, during and after the pandemic (2016, 2018, 2020, and 2021)—and how accurately can machine learning models predict temperature using real-time pollution data?

1 Methodology

1.1 Data Preprocessing

1.1.1 Data Collection

Data were collected from two different sources: air pollutant data from the U.S. Environmental Protection Agency (EPA) and weather data from a public open-source weather API named open-mateo dataset. We first concatenated individual pollutant datasets (four years for each pollutant, resulting in 16 datasets) to create one dataset per pollutant—O_3, CO, NO_2, and $PM_{2.5}$.

1.1.2 Data Merging

The four pollutant datasets were merged based on common attributes: date, county, latitude, and longitude. We retained data only for Bronx and Queens counties, which are geographically close enough to NYC to align with our weather dataset. The merged pollutant dataset was then joined with the weather dataset using the date feature to obtain the final dataset.

1.1.3 Handling Missing Values

Rows with missing weather data (173 in total) were removed to ensure data consistency and model readiness.

1.1.4 Feature Normalization

Min-Max normalization was applied to all numerical features to rescale values to the [0, 1] range, which improves model convergence and performance. The formula used for Min-Max scaling is:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

where X is the original feature value, $X_{\min}$ and $X_{\max}$ are the minimum and maximum values of the feature, and X' is the normalized value.

1.2 Data Visualization

1.2.1 Correlation Heatmap

From the figure below, we can see a clear correlation between ozone (i.e., O_3) and temperature, suggesting that ozone levels may influence temperature values. Moreover, there is noticeable correlation among the pollutants—for example, a correlation of nearly 0.7 between CO and NO_2. We also observe expected correlations among various weather factors, which aligns with our assumptions. Moreover, there is no correlation between any pollutant and rain precipitation - we will not include those label in our future ML model.

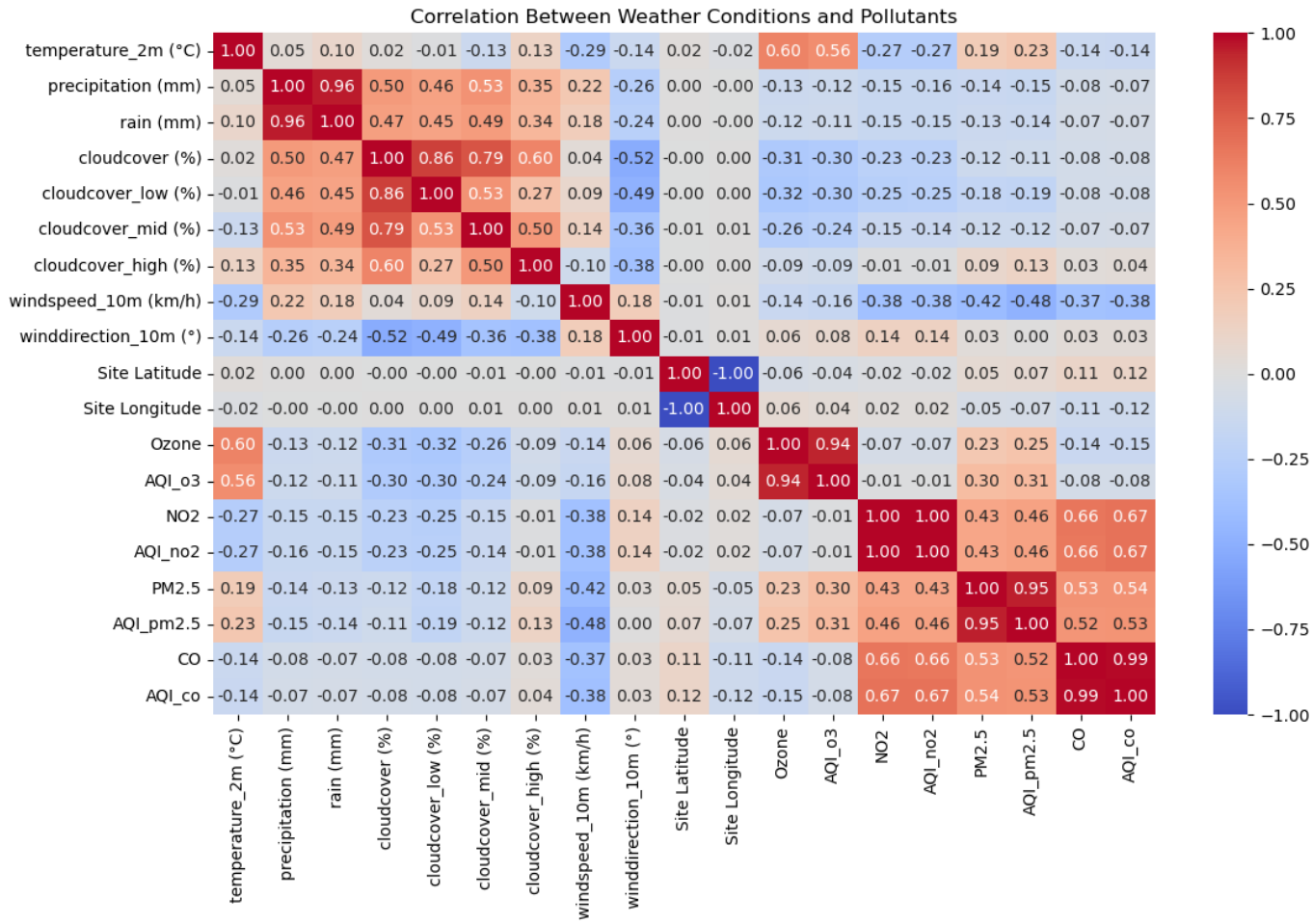

Figure 1: Correlation heatmap

1.2.2 Correlation Between Ozone and Temperature

In the figure below, we can observe a slight correlation between ozone and temperature. Specifically, when the temperature is low, ozone levels are also low, and as the temperature increases, ozone levels tend to rise as well

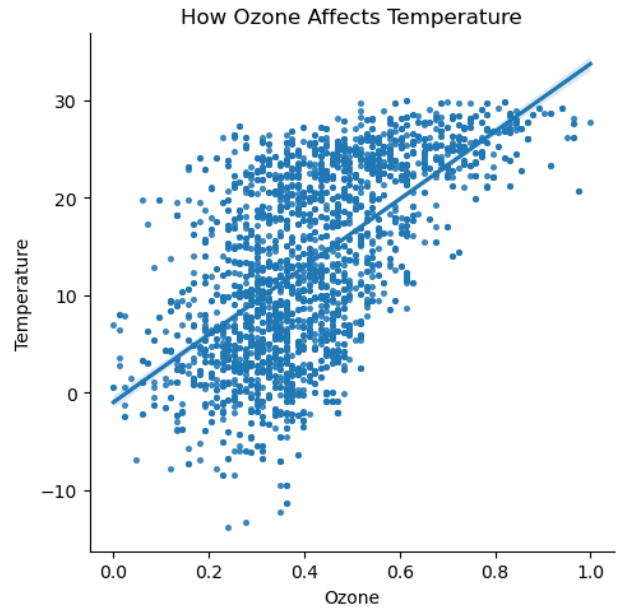

Figure 2: Correlation between ozone and temperature

1.2.3 Yearly Averages of Pollutants

The figure below clearly shows that the average value of each individual pollutant was at its lowest in 2020 (except O_3). This supports our earlier discussion and further suggests that pollution levels significantly dropped during that year (pick COVID year).

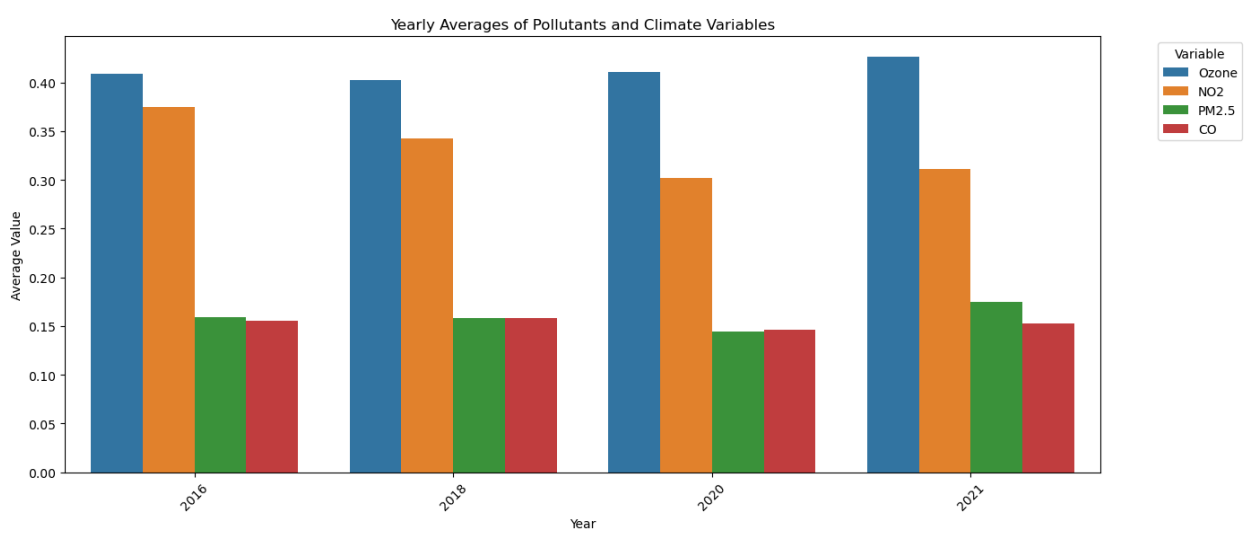

Figure 3: Yearly average concentrations of pollutants (O_3, CO, NO_2, and $PM_{2.5}$) in 2016, 2018, 2020 & 2021

1.3 Feature Engineering & Selection

1.3.1 One-Hot Encoding

One of the remaining issues in our dataset was the `Date` column, which was in a standard date format. To enable the model to learn from temporal patterns, we needed to convert this information into a numerical form.

We addressed this by extracting three temporal features—`Month`, `Day`, and `Year`—from the date. Each of these was then transformed using one-hot encoding, allowing the model to treat them as categorical variables with no inherent ordinal relationship.

For example:

- January (Month 1) → [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
- July (Month 7) → [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

This approach ensures that the model treats each month, day, and year as distinct categories rather than continuous values, thereby preventing misleading numerical relationships between them.

Next, we applied one-hot encoding to the `County` feature. As previously noted, our final dataset includes only two counties: Bronx and Queens. One-hot encoding converts this feature into two binary columns, ensuring that the model does not assume any ordinal relationship between the two locations.

1.3.2 KMeans Clustering for Latitude and Longitude

To transform the latitude and longitude features into more meaningful categorical variables, we applied KMeans clustering to the location data. We chose to cluster the data into two groups, which aligns with our dataset containing only two counties—Bronx and Queens. As expected, the clustering results reflected this separation, effectively encoding the geographic regions in a way that can be better understood by machine learning models.

1.3.3 Feature Selection

After applying one-hot encoding to the date and county features and clustering location data using KMeans, we were left with a total of 39 features and 2,896 rows.

We decided to exclude rain and precipitation from the set of target labels. As observed in the heatmap (see Figure 1), these variables showed high correlation with several other weather-related features, as well as with each other. Such multicollinearity can negatively impact model performance, particularly in regression tasks. Therefore, to ensure more robust and reliable predictions, we selected temperature as the sole label for our regression models.

Here's a polished and complete version of the **Model Building** subsection, including a clear description of your MLP model (VISL) and relevant mathematical definitions like ReLU activation and Linear layers:

1.4 Model Building

We built a simple Multi-Layer Perceptron (MLP) regression model, which we named **VISL** (Very Interpretable Structured Learner). This model is designed to predict temperature values based on the 39 input features described in the previous sections.

The VISL model architecture consists of several fully connected (dense) layers with non-linear activation and normalization applied in between. The model pipeline is as follows:

- A linear layer that maps the 39-dimensional input to 64 neurons.
- ReLU activation function:

$$\mathrm{ReLU}(x) = \max(0, x)$$

  This introduces non-linearity into the model and allows it to learn complex patterns.

- Batch Normalization, which normalizes the output of each layer to accelerate training and improve stability.
- A second linear layer projecting from 64 to 128 dimensions, followed by ReLU and Batch Normalization.
- A third linear layer from 128 to 64, again followed by ReLU and Batch Normalization.
- A fourth layer that reduces the dimensionality from 64 to 32, with ReLU and Batch Normalization.
- Finally, a single linear output neuron that predicts the target value (e.g., temperature)

Overall, the VISL model uses the following sequence of layers (implemented using PyTorch):

```
nn.Linear(39, 64)
nn.ReLU()
nn.BatchNorm1d(64)

nn.Linear(64, 128)
nn.ReLU()
nn.BatchNorm1d(128)

nn.Linear(128, 64)
nn.ReLU()
nn.BatchNorm1d(64)

nn.Linear(64, 32)
nn.ReLU()
nn.BatchNorm1d(32)

nn.Linear(32, 1)
```

The final layer outputs a continuous value suitable for regression tasks. We trained this model using the **Mean Absolute Error (MAE)** loss function, which is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i is the true value, $\hat{y}_i$ is the predicted value, and n is the number of data points. MAE measures the average magnitude of the errors without considering their direction, making it robust to outliers and easy to interpret.

The model was optimized using the **AdamW** optimizer, which is a variant of the Adam optimizer that decouples weight decay from the gradient update. After hyperparameter tuning, we selected a learning rate of 1×10^{-3} and a weight decay of 1×10^{-5}. The model was trained for 150 epochs.

2 Evaluation

2.0.1 Evaluation of VISL Model

To evaluate the performance of the VISL model, we rely on three commonly used regression metrics: **Mean Absolute Error (MAE)**, **Coefficient of Determination (R^2)**, and **Explained Variance Score (EVS)**. These metrics offer different perspectives on prediction accuracy and error spread. Their mathematical definitions and explanations are as follows:

- **Mean Absolute Error (MAE)**:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

MAE measures the average magnitude of errors between predicted and true values without considering their direction. It provides an easily interpretable error measure in the same units as the target variable.

- **Coefficient of Determination (R^2)**:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

  R^2 indicates the proportion of the variance in the target variable that is predictable from the input features. A score of 1 indicates perfect prediction, while 0 means the model does no better than the mean of the target.

- **Explained Variance Score (EVS)**:

$$\text{EVS} = 1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)}$$

  EVS measures the proportion to which a model accounts for the variation of the target variable. Higher values (closer to 1) indicate that the model explains more of the variability in the data.

To evaluate the robustness and consistency of the VISL model, we trained and tested it ten times using randomized data splits. In each run, the dataset was randomly divided into training, validation, and test sets, and a new instance of the model was initialized and trained from scratch.

This repeated training process allowed us to observe the stability of the model's performance under different data partitions. After each run, we recorded three key evaluation metrics—MAE, R^2, and EVS. By computing the mean, variance, and standard deviation of these metrics across all ten runs, we were able to assess the model's robustness and reliability more comprehensively than a single training instance would allow.

After training and evaluating the VISL model, we will compare its performance with two baseline models: **Linear Regression** and **Random Forest Regression**, to assess its effectiveness.

2.0.2 Comparison of VISL Performance with Existing Models

To evaluate the performance of the VISL model, we compared it against two well-known regression algorithms: ***Linear Regression*** and ***Random Forest Regressio***n. Each model was trained and evaluated 10 times using different randomized train/validation/test splits—consistent with the process used for ***VISL***. We computed three evaluation metrics for each model—MAE, R^2, and EVS—and recorded their **mean**, **variance**, and **standard deviation** across runs.

Linear Regression Statistics: {`'mae': [2.5144, 0.0, 0.0], 'r2': [0.8642, 0.0, 0.0], 'evs': [0.8643, 0.0, 0.0]`}

Random Forest Regression Statistics: {`'mae': [1.2258, 0.00034, 0.01855], 'r2': [0.9555, 0.0, 0.00107], 'evs': [0.9557, 0.0, 0.0011]`}

VISL Model Statistics: {`'mae': [1.384, 0.0078, 0.0886], 'r2': [0.9414, 0.0, 0.006], 'evs': [0.9433, 0.0, 0.0054]`}

2.0.3 Performance Summary

- The ***Linear Regression*** model showed the weakest performance, with a relatively high MAE of 2.5144 and zero variance across runs—indicating consistent but limited predictive capability.
- The ***Random Forest Regression*** model achieved the best results overall, with the lowest MAE (1.2258)

and highest R^2 (0.9555), demonstrating both strong accuracy and stability.
- The ***VISL*** model offered a competitive balance between performance and generalization. While its MAE (1.384) was slightly higher than Random Forest, it significantly outperformed Linear Regression. The moderate variance and standard deviation suggest VISL adapts well across different data splits and provides robust predictions.

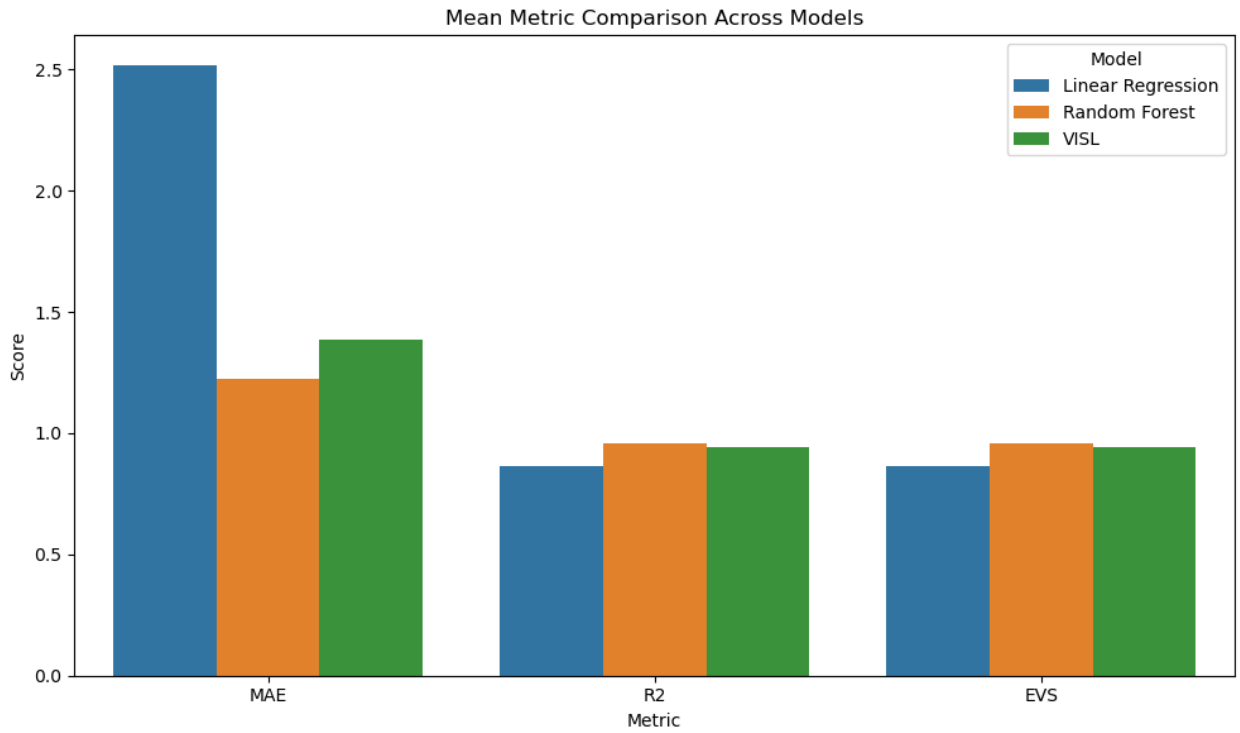

Figure 4: Bar plot comparing the average evaluation metric scores—Mean Absolute Error (MAE), Coefficient of Determination (R^2), and Explained Variance Score (EVS)—for the three models: Linear Regression, Random Forest Regression, and the VISL model. This visual highlights the superior performance of Random Forest across all metrics, with VISL closely following and outperforming Linear Regression.

3 Software/System Implementation

- **Programming Language:** The entire pipeline was implemented in Python 3.11, making use of its rich ecosystem of scientific and machine learning libraries.
- **Development Environment:** JupyterLab was used for development, experimentation, and debugging.
- **Version Control:** The project was fully managed with Git and hosted on GitHub to track changes and enable collaboration.
- **Reproducibility:** Random seeds were fixed throughout the pipeline to ensure reproducible results across different runs. Consistent data splits were maintained wherever applicable—except in the case of the 10 repeated model training runs, where different random splits were intentionally used without fixed seeds to assess robustness and generalization.
- **Modularity:** The codebase is organized under a well-structured `src/` directory, with clear separation between data preprocessing, model definition (VISL), training pipeline, evaluation scripts, and utility

functions. Datasets and dependency configuration files are also maintained separately to ensure clean and maintainable code.

- **Scalability & Efficiency:** Batch processing was applied during model training and evaluation to ensure memory efficiency and scalability across datasets of different sizes.
- **Availability:** All source code, dependencies, and datasets used in this project are publicly available in the GitHub repository: https://github.com/Gavision97/VISL_Project.git