Project Code

Applying Analytic Models on the WHO Ambient Air Quality Database

Team Name: ETA Bros

Discussion Section: DIS-06

Team Members:

- Justin Wang: justin17@stanford.edu
- Albert Tan: albert-tan@stanford.edu
- Peter Fu: peter107@stanford.edu
- Tianle Yao: tianle@stanford.edu

In [1]:
```
import json
import datetime
import requests
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import (
    mean_squared_error,
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    homogeneity_score,
    completeness_score,
    v_measure_score,
    adjusted_rand_score,
)
from sklearn.model_selection import (
    GridSearchCV,
    train_test_split,
    cross_val_score,
    cross_validate,
)
```

In [2]:
```
warnings.simplefilter(action="ignore", category=pd.errors.SettingWithCopyWarning)
```

In [3]:
```
df = pd.read_excel(
    "https://alberttan.github.io/static/sss/who_air_quality.xlsx", sheet_name=2
)
df.head()
```

Out[3]:

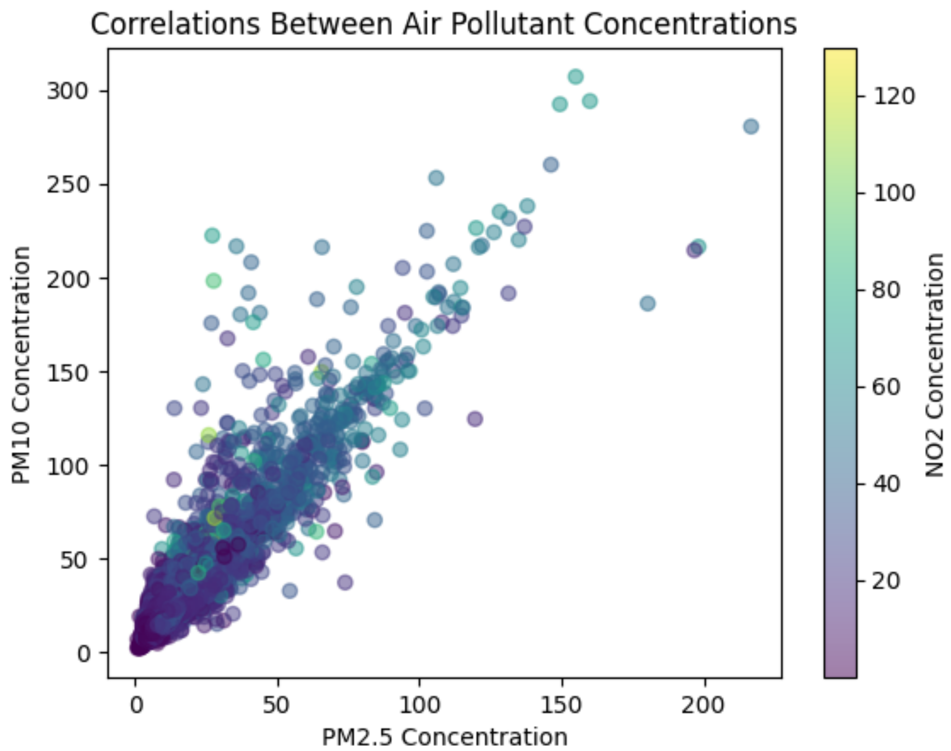
	who_region	iso3	country_name	city	year	version	pm10_concentration	pm25_concentrati
0	4_Eur	ESP	Spain	A Coruna/ ESP	2013.0	V4.0 (2018), V4.0 (2018), V4.0 (2018), V4.0 (2...	23.238	11.4
1	4_Eur	ESP	Spain	A Coruna/ ESP	2014.0	V6.0 (2023), V6.0 (2023), V6.0 (2023)	27.476	15.8
2	4_Eur	ESP	Spain	A Coruna/ ESP	2015.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	25.515	14.0
3	4_Eur	ESP	Spain	A Coruna/ ESP	2016.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	23.057	13.1
4	4_Eur	ESP	Spain	A Coruna/ ESP	2017.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	26.849	14.1

Question 1

Peter Fu

In [4]:

```
plt.scatter(
    df["pm25_concentration"],
    df["pm10_concentration"],
    c=df["no2_concentration"],
    alpha=0.5,
)
plt.title("Correlations Between Air Pollutant Concentrations")
plt.xlabel("PM2.5 Concentration")
plt.ylabel("PM10 Concentration")
plt.colorbar().set_label("NO2 Concentration")
plt.savefig("1.1.png", dpi=500)
```



In [5]:
```
df_q1_x = df[["pm25_concentration", "no2_concentration"]].fillna(0)
df_q1_y = df["pm10_concentration"].fillna(0)
distance = ["euclidean", "manhattan"]
X_train, X_test, y_train, y_test = train_test_split(
    df_q1_x, df_q1_y, test_size=0.1, random_state=12
)
```

In [6]:
```
pipeline_knn = make_pipeline(StandardScaler(), KNeighborsRegressor())

grid_cv = GridSearchCV(
    pipeline_knn,
    param_grid={
        "kneighborsregressor__n_neighbors": range(1, 40),
        "kneighborsregressor__metric": distance,
    },
    scoring="neg_mean_squared_error",
    cv=10,
)

grid_cv.fit(df_q1_x, df_q1_y)
```

Out[6]:
```
▸ GridSearchCV
▸ best_estimator_: Pipeline
    ▸ StandardScaler
    ▸ KNeighborsRegressor
```

In [7]:
```
grid_cv.best_params_
```

Out[7]:
```
{'kneighborsregressor__metric': 'euclidean',
 'kneighborsregressor__n_neighbors': 33}
```

In [8]:
```
pipeline_knn_optimized = make_pipeline(
    StandardScaler(), KNeighborsRegressor(n_neighbors=29)
)
```

In [9]:
```
scores = cross_val_score(
```

```python
    pipeline_knn_optimized,
    df_q1_x,
    df_q1_y,
    scoring="neg_mean_squared_error",
    cv=10,
)

(-scores.mean()), np.sqrt(-scores.mean())
```

Out[9]: (626.1399151224614, 25.022787916666307)

In [10…
```python
df_q1_y.std()
```

Out[10… 28.423565379629565

In [11…
```python
np.sqrt(-scores)
```

Out[11…
```
array([32.53519356, 20.80712393, 23.98856664, 27.39192813, 26.03645983,
       25.09041032, 22.77643582, 26.53455593, 22.05968204, 20.67004302])
```

In [12…
```python
pipeline_lr = make_pipeline(StandardScaler(), LinearRegression())

scores_lr = cross_val_score(
    pipeline_lr,
    df_q1_x,
    df_q1_y,
    scoring="neg_mean_squared_error",
    cv=10,
)
```

In [13…
```python
(-scores_lr.mean()), np.sqrt(-scores_lr.mean())
```

Out[13… (802.9923743530736, 28.337120078671962)

Question 2

Justin Wang

First, we will prepare our data by splitting `type_of_stations` into the appropriate columns `Urban`, `Suburban`, `Rural`, `Residential And Commercial Area`, `Urban Traffic/Residential And Commercial Area`, and `Urban Traffic` through the utilization of maps.

In [14…
```python
def transSplit(x):
    return x.split(", ")
```

In [15…
```python
rca = "Residential And Commercial Area"
utrca = "Urban Traffic/Residential And Commercial Area"
ut = "Urban Traffic"


def urban(x):
    if "Urban" in x:
        return pd.Series(x).value_counts(normalize=True).loc["Urban"]
    return 0.0


def suburban(x):
    if "Suburban" in x:
        return pd.Series(x).value_counts(normalize=True).loc["Suburban"]
    return 0.0


def rural(x):
    if "Rural" in x:
        return pd.Series(x).value_counts(normalize=True).loc["Rural"]
    return 0.0
```

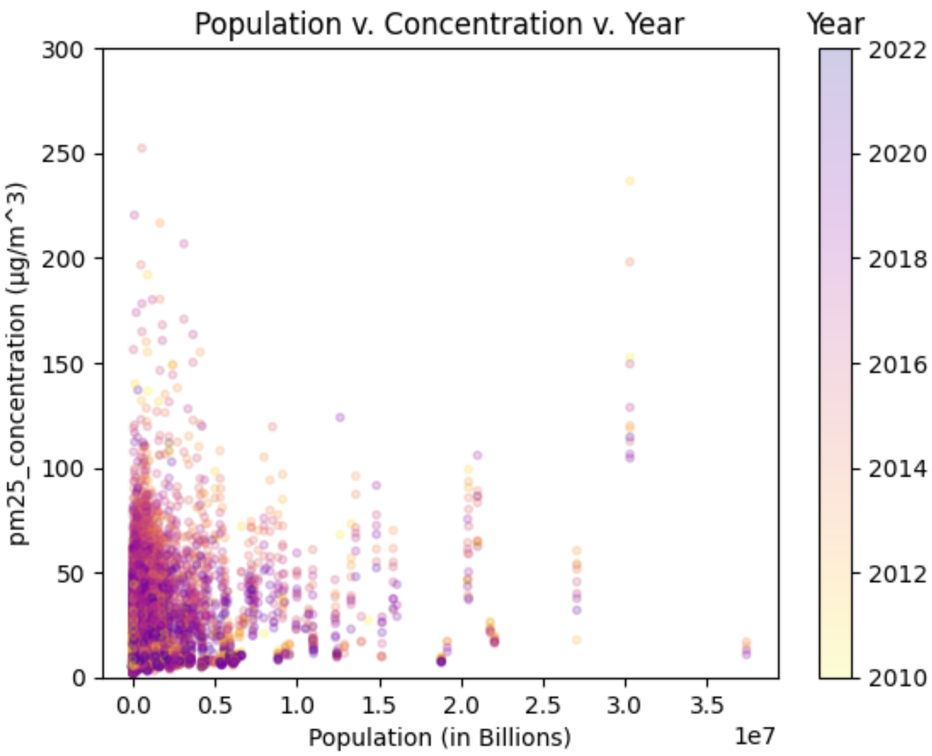
```python
def urban(x):
    if "Urban" in x:
        return pd.Series(x).value_counts(normalize=True).loc["Urban"]
    return 0.0


def RCA(x):
    if rca in x:
        return pd.Series(x).value_counts(normalize=True).loc[rca]
    return 0.0


def UTRCA(x):
    if utrca in x:
        return pd.Series(x).value_counts(normalize=True).loc[utrca]
    return 0.0


def UT(x):
    if ut in x:
        return pd.Series(x).value_counts(normalize=True).loc[ut]
    return 0.0
```

In [16…

```python
df_q2 = df.copy()
df_q2["type_of_stations"] = df["type_of_stations"].fillna("Undefined").map(transSplit)
df_q2["Urban"] = df_q2["type_of_stations"].map(urban)
df_q2["Suburban"] = df_q2["type_of_stations"].map(suburban)
df_q2["Rural"] = df_q2["type_of_stations"].map(rural)
df_q2[rca] = df_q2["type_of_stations"].map(RCA)
df_q2[utrca] = df_q2["type_of_stations"].map(UTRCA)
df_q2[ut] = df_q2["type_of_stations"].map(UT)
```

In [17…

```python
plt.scatter(
    df["population"],
    df["pm25_concentration"],
    alpha=0.2,
    s=10,
    c=df["year"],
    cmap="plasma_r",
)
plt.xlabel("Population (in Billions)")
plt.ylabel("pm25_concentration (µg/m^3)")
cb = plt.colorbar()
cb.ax.set_title("Year")
plt.title("Population v. Concentration v. Year")
plt.ylim(0, 300)
plt.savefig("2.1.png", dpi=500)
```



In [18…

```
df_q2[["type_of_stations", "Urban", "Suburban", "Rural", rca, utrca, ut]].head()
```

Out[18…

	type_of_stations	Urban	Suburban	Rural	Residential And Commercial Area	Urban Traffic/ Residential And Commercial Area	Urban Traffic
0	[Urban, Urban, Suburban]	0.666667	0.333333	0.0	0.0	0.0	0.0
1	[Urban, Urban, Suburban]	0.666667	0.333333	0.0	0.0	0.0	0.0
2	[Urban, Urban, Suburban, Suburban]	0.500000	0.500000	0.0	0.0	0.0	0.0
3	[Urban, Urban, Suburban, Suburban]	0.500000	0.500000	0.0	0.0	0.0	0.0
4	[Urban, Urban, Suburban, Suburban]	0.500000	0.500000	0.0	0.0	0.0	0.0

In [19…

```
ct = make_column_transformer(
    (StandardScaler(), ["year", "population", "Urban", "Suburban", rca, utrca, ut]),
    (OneHotEncoder(handle_unknown="ignore"), ["country_name"]),
    remainder="drop",
)
lr_model = make_pipeline(ct, LinearRegression())
knn_model = make_pipeline(ct, KNeighborsRegressor())
```

In [20…

```
df_q2 = df_q2[
    (~df_q2["population"].isnull())
    & (~df_q2["year"].isnull())
    & (~df_q2["pm25_concentration"].isnull())
]
```

In [21…

```python
lr_scores = cross_val_score(
    lr_model,
    df_q2,
    df_q2["pm25_concentration"],
    scoring="neg_root_mean_squared_error",
    cv=5,
)
RMSE_lr = -lr_scores.mean()
```

In [22…

```python
RMSE_lr
```

Out[22…

```
11.961090860966276
```

In [23…

```python
grid = GridSearchCV(
    knn_model,
    param_grid={
        "kneighborsregressor__n_neighbors": range(1, 10, 2),
        "kneighborsregressor__metric": ["euclidean", "manhattan"],
    },
    scoring="neg_root_mean_squared_error",
    cv=4,
)
grid.fit(df_q2, df_q2["pm25_concentration"])
```

Out[23…

```
▸ GridSearchCV
▸ best_estimator_: Pipeline
  ▸ columntransformer: ColumnTransformer
    ▸ standardscaler        ▸ onehotencoder
      ▸ StandardScaler        ▸ OneHotEncoder
  ▸ KNeighborsRegressor
```

In [24…

```python
grid.best_params_
```

Out[24…

```
{'kneighborsregressor__metric': 'manhattan',
 'kneighborsregressor__n_neighbors': 9}
```

In [25…

```python
pipe_knn = make_pipeline(ct, KNeighborsRegressor(metric="manhattan", n_neighbors=19))

pipe_knn.fit(df_q2, df_q2["pm25_concentration"])
```

Out[25…

```
▸ Pipeline
  ▸ columntransformer: ColumnTransformer
    ▸ standardscaler        ▸ onehotencoder
      ▸ StandardScaler        ▸ OneHotEncoder
  ▸ KNeighborsRegressor
```

In [26…

```python
knn_scores = cross_val_score(
    pipe_knn,
    df_q2,
    df_q2["pm25_concentration"],
    scoring="neg_root_mean_squared_error",
    cv=5,
)
```

In [27…
```
RMSE_knn = -knn_scores.mean()
```

In [28…
```
df_q2["pm25_concentration"].max() - df_q2["pm25_concentration"].min()
```

Out[28… 434.662

In [29…
```
RMSE_lr
```

Out[29… 11.961090860966276

In [30…
```
RMSE_knn
```

Out[30… 11.925448330768308

In [31…
```
pd.DataFrame(
    {"Model": ["Linear Regression", "KNN"], "RMSE": [RMSE_lr, RMSE_knn]}
).set_index("Model")
```

Out[31…

	RMSE
Model	
Linear Regression	11.961091
KNN	11.925448

In [32…
```
lr_model.fit(df_q2, df_q2["pm25_concentration"])
plt.scatter(df_q2["pm25_concentration"], lr_model.predict(df_q2), alpha=0.2, s=10)
plt.plot(range(0, 400), range(0, 400), c="red", alpha=0.5)
plt.ylim(0, 120)
plt.xlim(0, 120)
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("Linear Regression Training Model Accuracy")
plt.savefig("2.2.png", dpi=500)
```

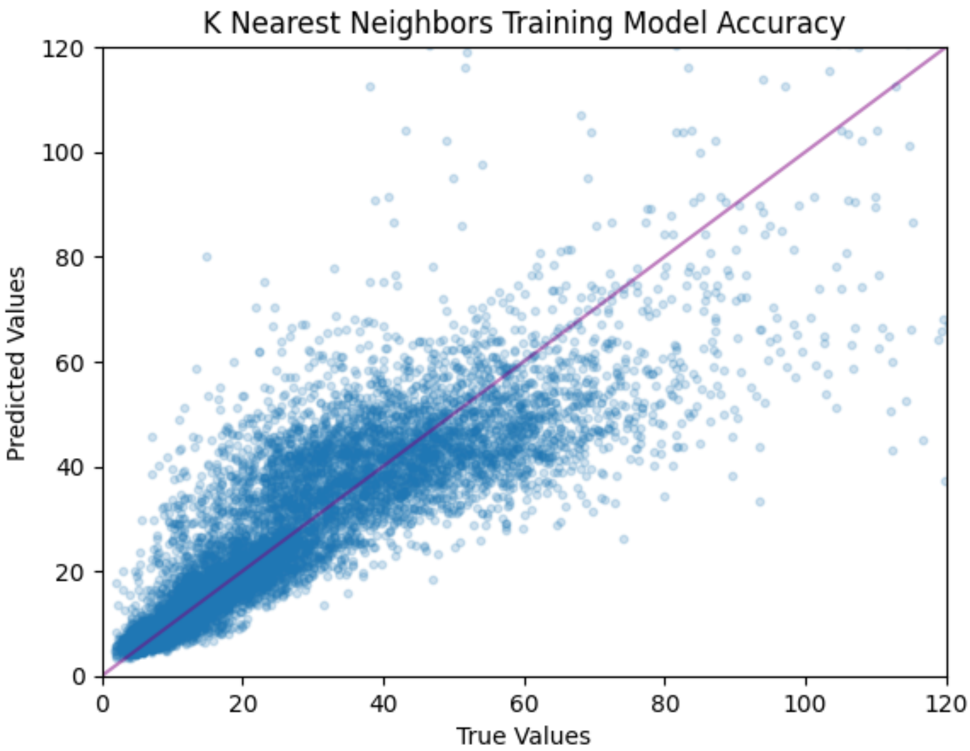
In [33…
```
np.sqrt(mean_squared_error(df_q2["pm25_concentration"], lr_model.predict(df_q2)))
```

Out[33… 11.582749477681292

In [34…
```
np.sqrt(mean_squared_error(df_q2["pm25_concentration"], pipe_knn.predict(df_q2)))
```

Out[34… 10.62365459929716

In [35…
```
knn_model.fit(df_q2, df_q2["pm25_concentration"])
plt.scatter(df_q2["pm25_concentration"], knn_model.predict(df_q2), alpha=0.2, s=10)
plt.plot(range(0, 400), range(0, 400), c="purple", alpha=0.5)
plt.ylim(0, 120)
plt.xlim(0, 120)
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("K Nearest Neighbors Training Model Accuracy")
plt.savefig("2.3.png", dpi=500)
```



Question 3

Albert Tan

Data Refinement & Analysis

In our dataset, the same location might give multiple entries in different years. If we include the data of all years, the nearest neighbors of a location might be another entry of the location itself. For our analysis to be more useful, we select a year to focus our study.

In [36…
```
pd.DataFrame(
    [
        df.groupby("year")["country_name"].apply(len),
        df.groupby("year")["country_name"].unique().apply(len),
    ],
    index=["Number of Entries", "Unique Countries"],
).transpose()
```

Out[36...

year	Number of Entries	Unique Countries
2010.0	2878	65
2011.0	802	37
2012.0	971	41
2013.0	3869	77
2014.0	3460	80
2015.0	4096	84
2016.0	4131	81
2017.0	3820	79
2018.0	4678	86
2019.0	4594	100
2020.0	3983	73
2021.0	2654	48
2022.0	159	10

2019 has the second highest number of entries and the most number of unique countries. Thus we use the data of the year 2019 to conduct our analysis. Moreover, for us to use K Nearest Neighbors, countries with too few locations should be removed since no appropriate neighbors can be found for their locations.

In [37...

```
df_2019 = df[df["year"] == 2019]
df_q3 = df_2019[
    df_2019["country_name"].apply(
        lambda x: df_2019["country_name"].value_counts()[x] >= 3
    )
].copy()
df_q3_discarded = df_2019[
    df_2019["country_name"].apply(
        lambda x: df_2019["country_name"].value_counts()[x] < 3
    )
].copy()
df_q3.head()
```

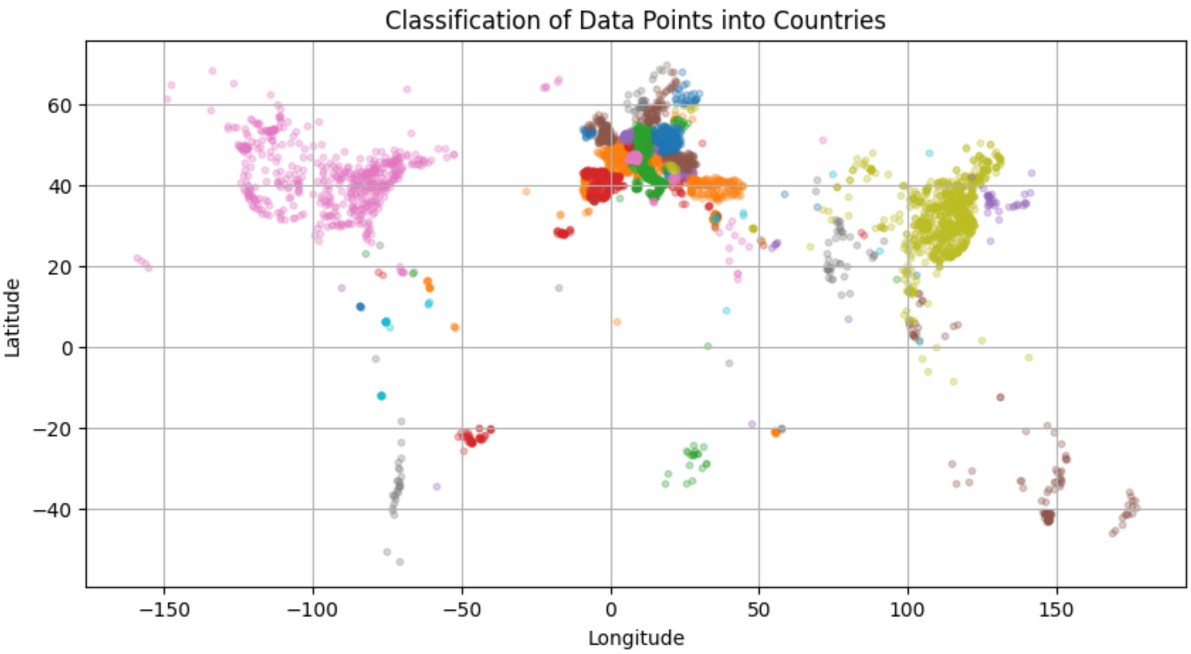
Out[37…

	who_region	iso3	country_name	city	year	version	pm10_concentration	pm25_concentra
6	4_Eur	ESP	Spain	A Coruna/ ESP	2019.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	26.912	1:
18	4_Eur	DEU	Germany	Aachen/ DEU	2019.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	16.090	8
27	4_Eur	CHE	Switzerland	Aadorf/ CHE	2019.0	V5.0 (2022), V5.0 (2022)	10.931	
35	4_Eur	DNK	Denmark	Aalborg/ DNK	2019.0	V6.0 (2023)	NaN	9
44	4_Eur	DEU	Germany	Aalen/ DEU	2019.0	V6.0 (2023)	13.862	

A scatterplot can be used to visually display how all data in 2019 can be categorized based on countries.

In [38…

```
plt.figure(figsize=(10, 5))
df_2019.groupby("country_name").apply(
    lambda x: plt.scatter(x=x["longitude"], y=x["latitude"], s=10, alpha=0.3),
    include_groups=False,
)
plt.title("Classification of Data Points into Countries")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid()
plt.savefig("3.1.png", dpi=500)
```



In [39…

```
grid_cv = GridSearchCV(
    KNeighborsClassifier(),
    param_grid={"n_neighbors": range(1, 31)},
    scoring="accuracy",
```

```
    cv=10,
    return_train_score=True,
)
```

In [40…]

```
grid_cv.fit(df_q3[["longitude", "latitude"]], df_q3["country_name"])
```

```
/Users/albert/Library/Python/3.12/lib/python/site-packages/sklearn/model_selection/_spl
it.py:776: UserWarning: The least populated class in y has only 3 members, which is les
s than n_splits=10.
  warnings.warn(
```

Out[40…]

```
▸ GridSearchCV
▸ best_estimator_: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

In [41…]

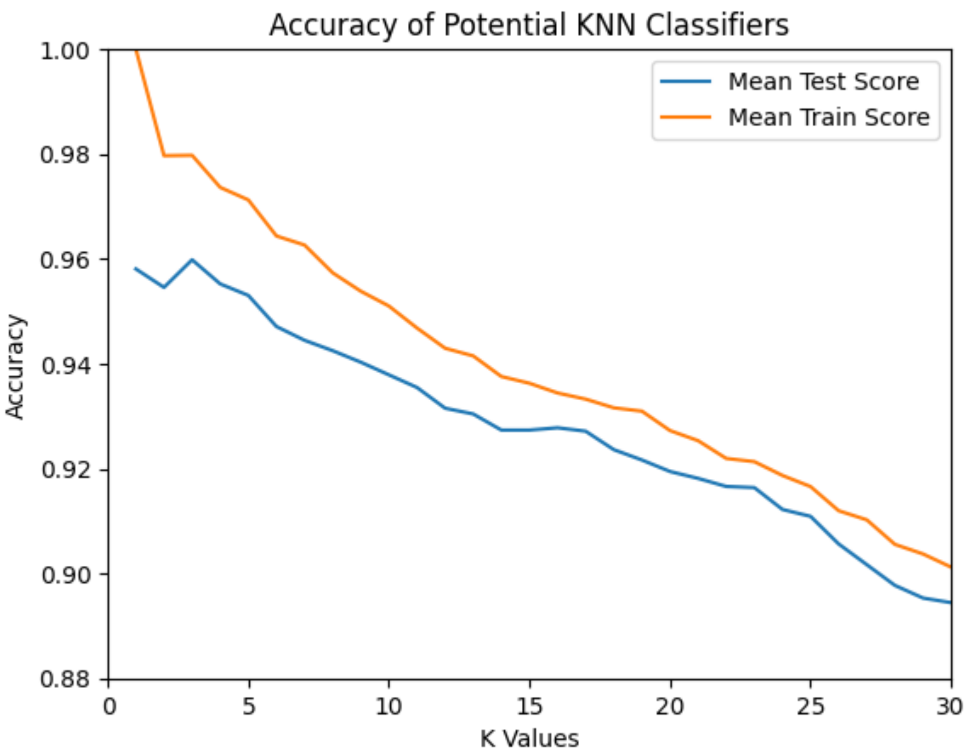
```
grid_cv.best_params_, grid_cv.cv_results_["mean_test_score"].max()
```

Out[41…]

```
({'n_neighbors': 3}, 0.9598332369385002)
```

In [42…]

```
plt.plot(range(1, 31), grid_cv.cv_results_["mean_test_score"], label="Mean Test Score")
plt.plot(
    range(1, 31), grid_cv.cv_results_["mean_train_score"], label="Mean Train Score"
)
plt.title("Accuracy of Potential KNN Classifiers")
plt.xlabel("K Values")
plt.ylabel("Accuracy")
plt.xlim(0, 30)
plt.ylim(0.88, 1)
plt.legend()
plt.savefig("3.2.png", dpi=500)
```



In [43…]

```
knn_q3 = KNeighborsClassifier(n_neighbors=3)
knn_q3.fit(df_q3[["longitude", "latitude"]], df_q3["country_name"])
knn_q3.predict(df_q3[["longitude", "latitude"]])
```

Out[43…]

```
array(['Spain', 'Germany', 'Switzerland', ..., 'Poland',
       'Republic of Korea', 'China'], dtype=object)
```

In [44…]

```
df_q3["country_consistency"] = df_q3["country_name"] == knn_q3.predict(
```

```
    df_q3[["longitude", "latitude"]]
)
df_q3.head()
```

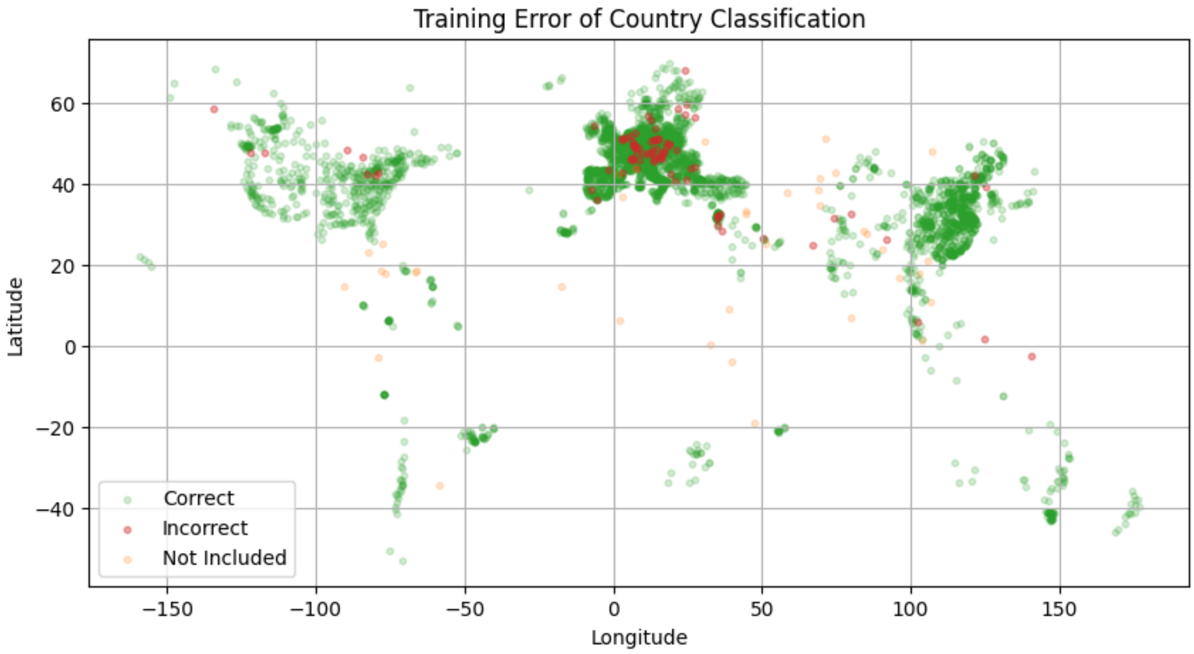
Out[44…

	who_region	iso3	country_name	city	year	version	pm10_concentration	pm25_concentra
6	4_Eur	ESP	Spain	A Coruna/ ESP	2019.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	26.912	1:
18	4_Eur	DEU	Germany	Aachen/ DEU	2019.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	16.090	8
27	4_Eur	CHE	Switzerland	Aadorf/ CHE	2019.0	V5.0 (2022), V5.0 (2022)	10.931	
35	4_Eur	DNK	Denmark	Aalborg/ DNK	2019.0	V6.0 (2023)	NaN	9
44	4_Eur	DEU	Germany	Aalen/ DEU	2019.0	V6.0 (2023)	13.862	

5 rows × 21 columns

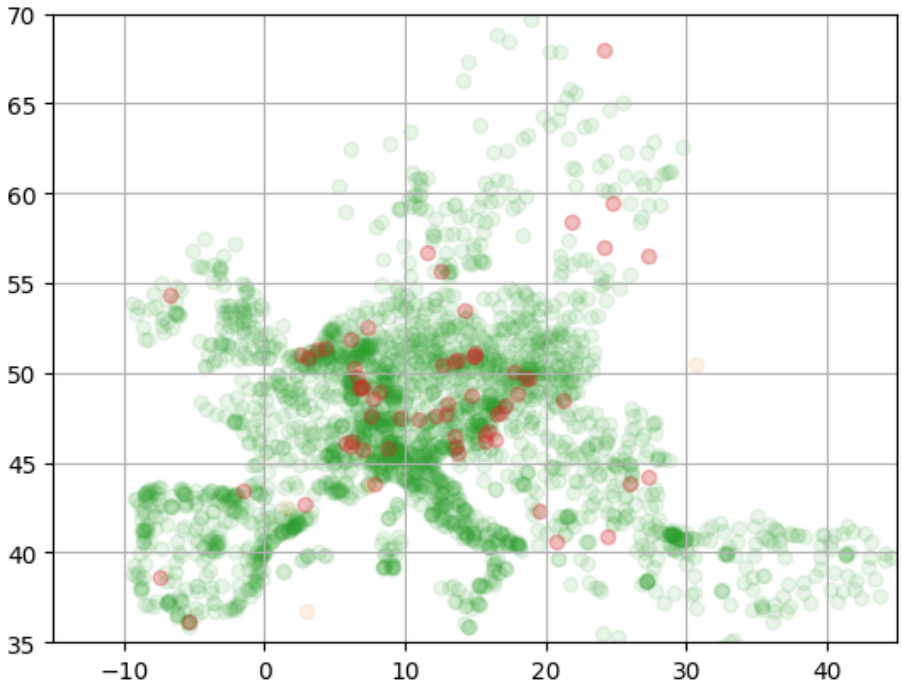
In [45…

```
plt.figure(figsize=(10, 5))
plt.scatter(
    x=df_q3[df_q3["country_consistency"] == True]["longitude"],
    y=df_q3[df_q3["country_consistency"] == True]["latitude"],
    c="tab:green",
    s=10,
    alpha=0.2,
    label="Correct",
)
plt.scatter(
    x=df_q3[df_q3["country_consistency"] == False]["longitude"],
    y=df_q3[df_q3["country_consistency"] == False]["latitude"],
    c="tab:red",
    s=10,
    alpha=0.4,
    label="Incorrect",
)
plt.scatter(
    x=df_q3_discarded["longitude"],
    y=df_q3_discarded["latitude"],
    c="tab:orange",
    s=10,
    alpha=0.2,
    label="Not Included",
)
# df_2019.groupby("who_region_consistency").apply(lambda x: plt.scatter(x=x["longitude"
plt.title("Training Error of Country Classification")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.legend()
plt.grid()
plt.savefig("3.3.png", dpi=500)
```



In [46…

```
plt.scatter(
    x=df_q3[df_q3["country_consistency"] == True]["longitude"],
    y=df_q3[df_q3["country_consistency"] == True]["latitude"],
    c="tab:green",
    alpha=0.1,
)
plt.scatter(
    x=df_q3[df_q3["country_consistency"] == False]["longitude"],
    y=df_q3[df_q3["country_consistency"] == False]["latitude"],
    c="tab:red",
    alpha=0.3,
)
plt.scatter(
    x=df_q3_discarded["longitude"],
    y=df_q3_discarded["latitude"],
    c="tab:orange",
    alpha=0.1,
)
plt.grid()
plt.xlim(-15, 45)
plt.ylim(35, 70)
plt.show()
```



In [47…

```
pd.DataFrame(
    confusion_matrix(
        df_q3["country_name"], knn_q3.predict(df_q3[["longitude", "latitude"]])
    ),
    index=knn_q3.classes_,
    columns=knn_q3.classes_,
)
```

Out[47…

	Albania	Australia	Austria	Bahrain	Belgium	Bosnia and Herzegovina	Brazil	Bulgaria	Cambodia
Albania	3	0	0	0	0	0	0	0	0
Australia	0	68	0	0	0	0	0	0	0
Austria	0	0	112	0	0	0	0	0	0
Bahrain	0	0	0	3	0	0	0	0	0
Belgium	0	0	0	0	57	0	0	0	0
...	...	...	...	...	...	...	...	...	...
T√°rkiye	0	0	0	0	0	0	0	0	0
United Arab Emirates	0	0	0	0	0	0	0	0	0
United Kingdom of Great Britain and Northern Ireland	0	0	0	0	0	0	0	0	0
United States of America	0	0	0	0	0	0	0	0	0
occupied Palestinian territory, including east Jerusalem	0	0	0	0	0	0	0	0	0

67 rows × 67 columns

In [48…

```
accuracy_score(df_q3["country_name"], knn_q3.predict(df_q3[["longitude", "latitude"]]))
```

Out[48… 0.9804653204565408

Question 4

Albert Tan

Data Refinement & Analysis

In [49…

```
df_q4 = df.dropna(
    subset=["pm25_concentration", "pm10_concentration", "no2_concentration"]
)

file = requests.get(
    "https://albertttan.github.io/static/sss/country_classification.json"
).text
countries_raw = json.loads(file)
countries = {}
for key in countries_raw.keys():
    for value in countries_raw[key]:
        if key == "2":
            countries[value] = 1
        else:
```

```python
            countries[value] = 0
df_q4["development"] = df_q4["country_name"].map(countries)

for metric in ["pm25", "pm10", "no2"]:
    df_q4[metric + "_standardized"] = StandardScaler().fit_transform(
        df_q4[[metric + "_concentration"]]
    )

df_q4.head()
```

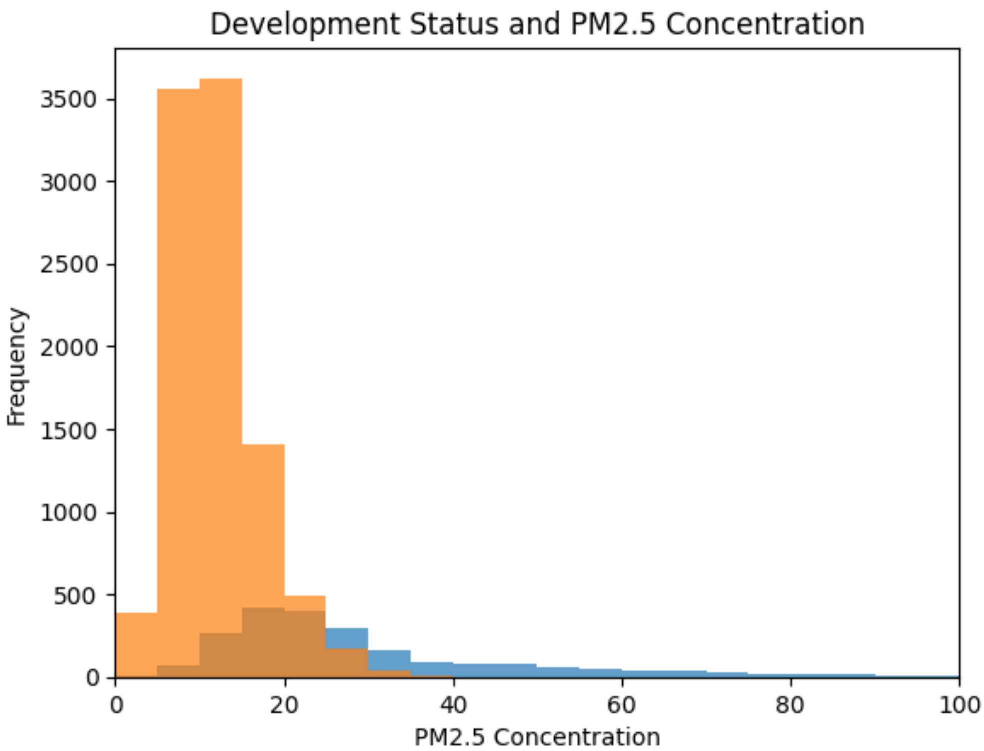
Out[49…

	who_region	iso3	country_name	city	year	version	pm10_concentration	pm25_concentrati
0	4_Eur	ESP	Spain	A Coruna/ ESP	2013.0	V4.0 (2018), V4.0 (2018), V4.0 (2018), V4.0 (2...	23.238	11.4
1	4_Eur	ESP	Spain	A Coruna/ ESP	2014.0	V6.0 (2023), V6.0 (2023), V6.0 (2023)	27.476	15.8
2	4_Eur	ESP	Spain	A Coruna/ ESP	2015.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	25.515	14.0
3	4_Eur	ESP	Spain	A Coruna/ ESP	2016.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	23.057	13.1
4	4_Eur	ESP	Spain	A Coruna/ ESP	2017.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	26.849	14.1

5 rows × 24 columns

In [50…

```python
df_q4.groupby("development")["pm25_concentration"].plot.hist(
    bins=np.linspace(0, 100, 21), alpha=0.7
)
plt.title("Development Status and PM2.5 Concentration")
plt.xlabel("PM2.5 Concentration")
plt.xlim(0, 100)
plt.savefig("4.4.png", dpi=500)
```



Supervised Learning

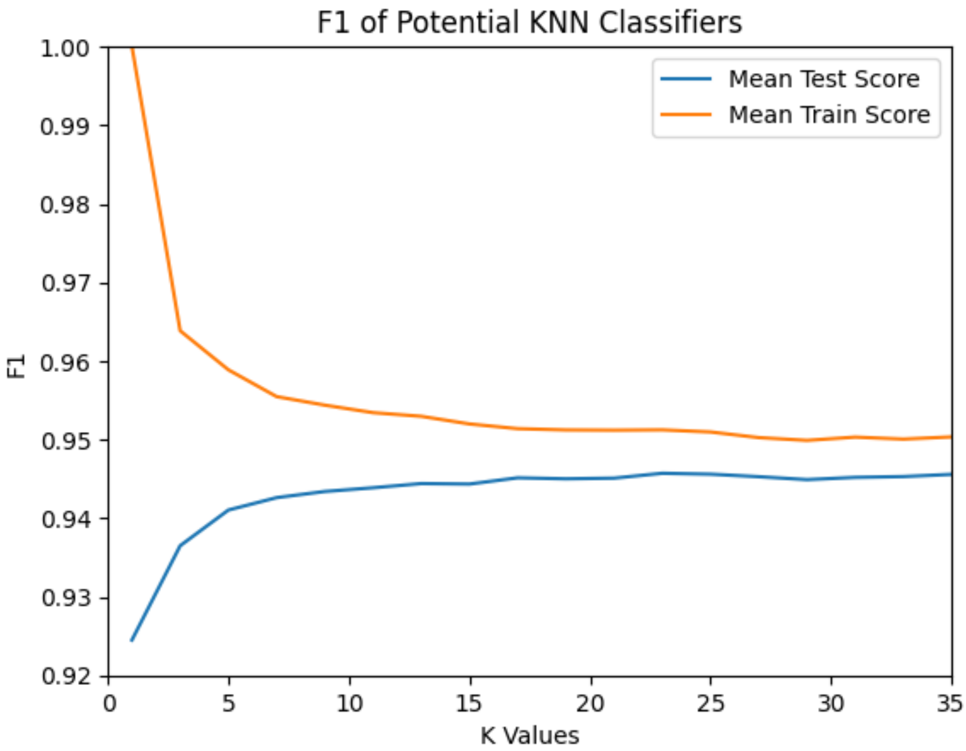
In [51…
```
choices = {
    "PM2.5": ["pm25_standardized"],
    "PM10": ["pm10_standardized"],
    "NO2": ["no2_standardized"],
    "PM2.5 + PM10": ["pm25_standardized", "pm10_standardized"],
    "PM2.5 + NO2": ["pm25_standardized", "no2_standardized"],
    "PM10 + NO2": ["pm10_standardized", "no2_standardized"],
    "PM2.5 + PM10 + NO2": [
        "pm25_standardized",
        "pm10_standardized",
        "no2_standardized",
    ],
}
logreg_q4 = LogisticRegression()
```

In [52…
```
grid_cv_knn_q4 = GridSearchCV(
    KNeighborsClassifier(),
    param_grid={"n_neighbors": range(1, 36, 2)},
    scoring="f1",
    cv=5,
    return_train_score=True,
)
grid_cv_knn_q4.fit(
    df_q4[["pm25_standardized", "pm10_standardized", "no2_standardized"]],
    df_q4["development"],
)
grid_cv_knn_q4.best_params_, grid_cv_knn_q4.cv_results_["mean_test_score"].max()
```

Out[52…
```
({'n_neighbors': 23}, 0.9457128707975235)
```

In [53…
```
knn_q4 = KNeighborsClassifier(n_neighbors=grid_cv_knn_q4.best_params_["n_neighbors"])
plt.plot(
    range(1, 36, 2),
    grid_cv_knn_q4.cv_results_["mean_test_score"],
    label="Mean Test Score",
)
plt.plot(
    range(1, 36, 2),
    grid_cv_knn_q4.cv_results_["mean_train_score"],
```

```python
    label="Mean Train Score",
)
plt.title("F1 of Potential KNN Classifiers")
plt.xlabel("K Values")
plt.ylabel("F1")
plt.xlim(0, 35)
plt.ylim(0.92, 1)
plt.legend()
plt.savefig("4.1.png", dpi=500)
```



In [54...

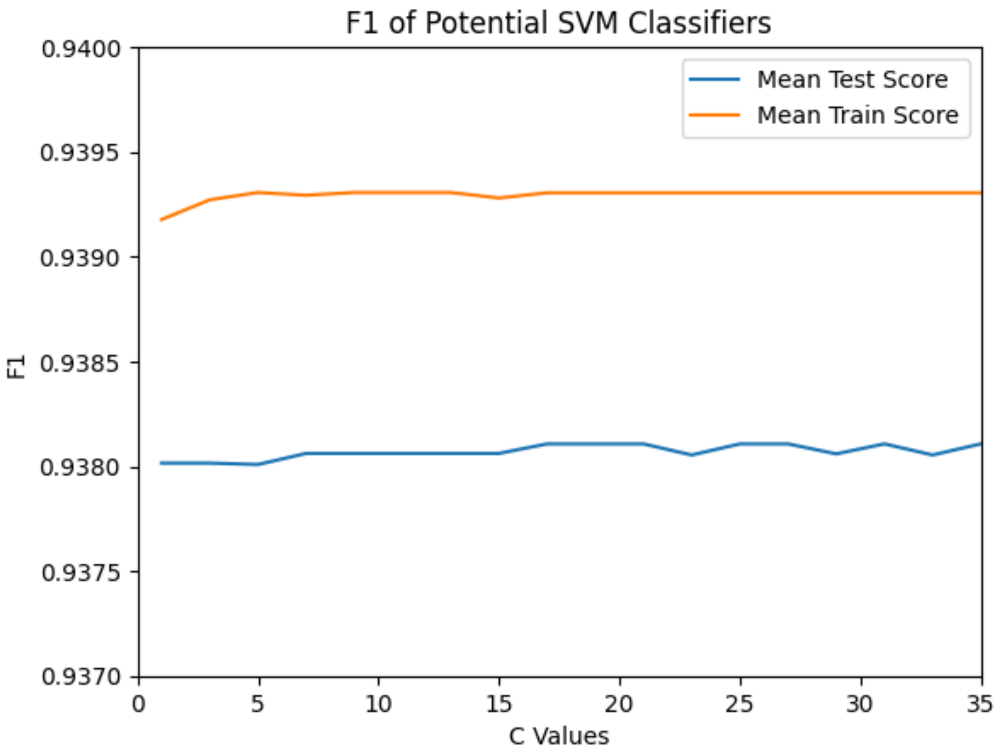
```python
grid_cv_svm_q4 = GridSearchCV(
    SVC(kernel="linear"),
    param_grid={"C": range(1, 36, 2)},
    scoring="f1",
    cv=5,
    return_train_score=True,
)
grid_cv_svm_q4.fit(
    df_q4[["pm25_standardized", "pm10_standardized", "no2_standardized"]],
    df_q4["development"],
)
grid_cv_svm_q4.best_params_, grid_cv_svm_q4.cv_results_["mean_test_score"].max()
```

Out[54...

```
({'C': 17}, 0.9381067314486449)
```

In [55…

```
svm_q4 = SVC(kernel="linear", C=grid_cv_svm_q4.best_params_["C"])
plt.plot(
    range(1, 36, 2),
    grid_cv_svm_q4.cv_results_["mean_test_score"],
    label="Mean Test Score",
)
plt.plot(
    range(1, 36, 2),
    grid_cv_svm_q4.cv_results_["mean_train_score"],
    label="Mean Train Score",
)
plt.title("F1 of Potential SVM Classifiers")
plt.xlabel("C Values")
plt.ylabel("F1")
plt.xlim(0, 35)
plt.ylim(0.937, 0.940)
plt.legend()
plt.savefig("4.2.png", dpi=500)
```



In [56…

```
df_f1 = pd.DataFrame(index=choices.keys(), columns=["LogReg", "KNN", "SVM"])

for index in choices.keys():
    choice = choices[index]
    accuracy_logreg = cross_val_score(
        logreg_q4, X=df_q4[choice], y=df_q4["development"], scoring="f1", cv=5
    )
    accuracy_knn = cross_val_score(
        knn_q4, X=df_q4[choice], y=df_q4["development"], scoring="f1", cv=5
    )
    accuracy_svm = cross_val_score(
        svm_q4, X=df_q4[choice], y=df_q4["development"], scoring="f1", cv=5
    )
    df_f1.loc[index, "LogReg"] = accuracy_logreg.mean()
    df_f1.loc[index, "KNN"] = accuracy_knn.mean()
    df_f1.loc[index, "SVM"] = accuracy_svm.mean()

df_f1.head()
```

Out[56...

	LogReg	KNN	SVM
PM2.5	0.931671	0.929827	0.930936
PM10	0.94007	0.939616	0.939884
NO2	0.904468	0.901297	0.899035
PM2.5 + PM10	0.93807	0.937136	0.938145
PM2.5 + NO2	0.931094	0.934049	0.930189

In [57...

```
# Second best model

logreg_q4.fit(df_q4[choices["PM10"]], df_q4["development"])

print(
    "Developing F1:",
    f1_score(
        df_q4["development"], logreg_q4.predict(df_q4[choices["PM10"]]), pos_label=0
    ),
)
print(
    "Developed F1:",
    f1_score(
        df_q4["development"], logreg_q4.predict(df_q4[choices["PM10"]]), pos_label=1
    ),
)
pd.DataFrame(
    confusion_matrix(df_q4["development"], logreg_q4.predict(df_q4[choices["PM10"]])),
    index=["Developing", "Developed"],
    columns=["Developing", "Developed"],
)
```

```
Developing F1: 0.6664830625172129
Developed F1: 0.9397422500870777
```

Out[57...

	Developing	Developed
Developing	1210	966
Developed	245	9443

In [58...

```
# Best model

knn_q4.fit(df_q4[choices["PM2.5 + PM10 + NO2"]], df_q4["development"])

print(
    "Developing F1:",
    f1_score(
        df_q4["development"],
        knn_q4.predict(df_q4[choices["PM2.5 + PM10 + NO2"]]),
        pos_label=0,
    ),
)
print(
    " Developed F1:",
    f1_score(
        df_q4["development"],
        knn_q4.predict(df_q4[choices["PM2.5 + PM10 + NO2"]]),
        pos_label=1,
    ),
)
pd.DataFrame(
    confusion_matrix(
        df_q4["development"], knn_q4.predict(df_q4[choices["PM2.5 + PM10 + NO2"]])
    ),
    index=["Developing", "Developed"],
    columns=["Developing", "Developed"],
)
```

```
Developing F1: 0.7346609257265877
 Developed F1: 0.9507295622626424
```

Out[58…

	Developing	Developed
Developing	1365	811
Developed	175	9513

Unsupervised Learning

In [59…

```
kmeans = KMeans(n_clusters=2, init="random", n_init=1)
kmeans.fit(df_q4[["pm25_standardized", "pm10_standardized", "no2_standardized"]])
df_q4["label"] = pd.Series(kmeans.labels_, index=df_q4.index)
df_q4.head()
```

Out[59…

	who_region	iso3	country_name	city	year	version	pm10_concentration	pm25_concentrati
0	4_Eur	ESP	Spain	A Coruna/ ESP	2013.0	V4.0 (2018), V4.0 (2018), V4.0 (2018), V4.0 (2...	23.238	11.4
1	4_Eur	ESP	Spain	A Coruna/ ESP	2014.0	V6.0 (2023), V6.0 (2023), V6.0 (2023)	27.476	15.8
2	4_Eur	ESP	Spain	A Coruna/ ESP	2015.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	25.515	14.0
3	4_Eur	ESP	Spain	A Coruna/ ESP	2016.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	23.057	13.1
4	4_Eur	ESP	Spain	A Coruna/ ESP	2017.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	26.849	14.1

5 rows × 25 columns

In [60…

```
group0 = df_q4.groupby("label")["pm25_concentration"].mean().sort_values().index[0]
group1 = df_q4.groupby("label")["pm25_concentration"].mean().sort_values().index[1]
name_map = {group0: "low", group1: "high"}
color_map = {"low": "tab:green", "high": "tab:red"}
```

In [61…

```
df_q4["label"] = df_q4["label"].map(name_map)
df_q4.head()
```

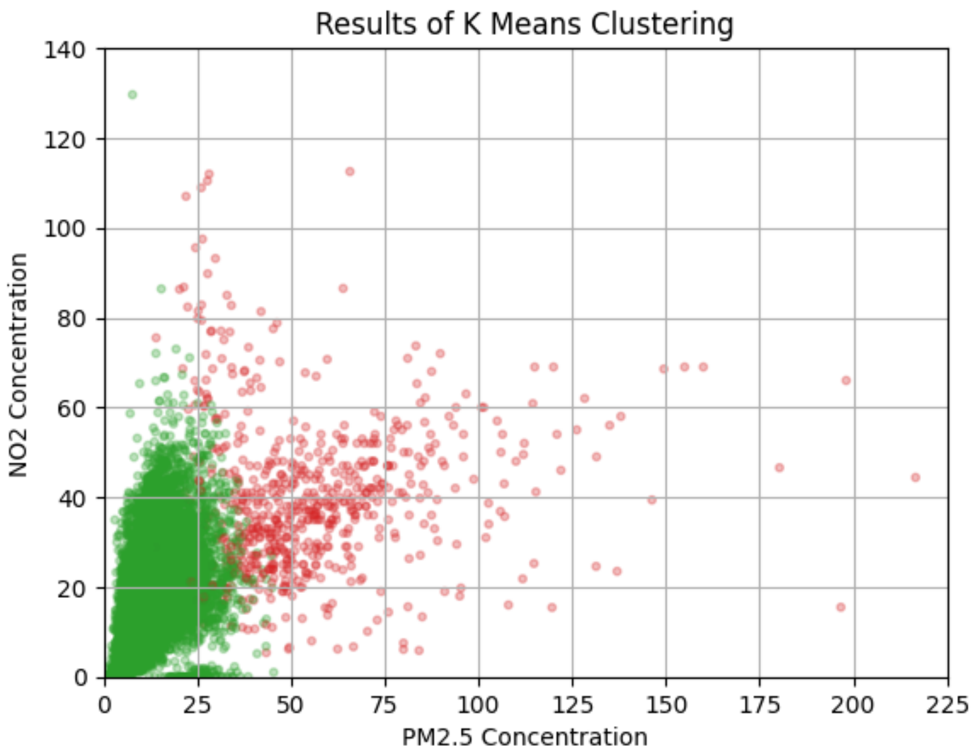
Out[61...

	who_region	iso3	country_name	city	year	version	pm10_concentration	pm25_concentrati
0	4_Eur	ESP	Spain	A Coruna/ ESP	2013.0	V4.0 (2018), V4.0 (2018), V4.0 (2018), V4.0 (2...	23.238	11.4
1	4_Eur	ESP	Spain	A Coruna/ ESP	2014.0	V6.0 (2023), V6.0 (2023), V6.0 (2023)	27.476	15.8
2	4_Eur	ESP	Spain	A Coruna/ ESP	2015.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	25.515	14.0
3	4_Eur	ESP	Spain	A Coruna/ ESP	2016.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	23.057	13.1
4	4_Eur	ESP	Spain	A Coruna/ ESP	2017.0	V6.0 (2023), V6.0 (2023), V6.0 (2023), V6.0...	26.849	14.1

5 rows × 25 columns

In [62...

```
plt.scatter(
    x=df_q4["pm25_concentration"],
    y=df_q4["no2_concentration"],
    c=df_q4["label"].map(color_map),
    s=10,
    alpha=0.3,
)
plt.title("Results of K Means Clustering")
plt.xlabel("PM2.5 Concentration")
plt.ylabel("NO2 Concentration")
plt.xlim(0, 225)
plt.ylim(0, 140)
plt.grid()
plt.savefig("4.3.png", dpi=500)
```



In [63…

```
def cluster_purity(y, clusters, cluster_label):
    idx = clusters == cluster_label
    classes = y[idx]
    purity = classes.value_counts(normalize=True).max()
    return purity


def clustering_purity(y, clusters):
    clSeries = pd.Series(clusters)
    sizes = clSeries.value_counts().sort_index()
    cluster_labels = sizes.index
    purities = pd.Series({l: cluster_purity(y, clusters, l) for l in cluster_labels})
    prepped = pd.DataFrame({"Purity": purities, "Size": sizes})
    n = len(y)
    purity = (prepped.Purity * prepped.Size).sum() / n
    return purity
```

In [64…

```
pd.DataFrame(
    [
        clustering_purity(
            df_q4["development"], df_q4["label"].map({"low": 1, "high": 0})
        ),
        v_measure_score(df_q4["development"], df_q4["label"]),
        adjusted_rand_score(df_q4["development"], df_q4["label"]),
    ],
    index=["Purity", "V Measure", "Adjusted Rand"],
    columns=["Scores"],
)
```

Out[64…

	Scores
Purity	0.870027
V Measure	0.283009
Adjusted Rand	0.335486

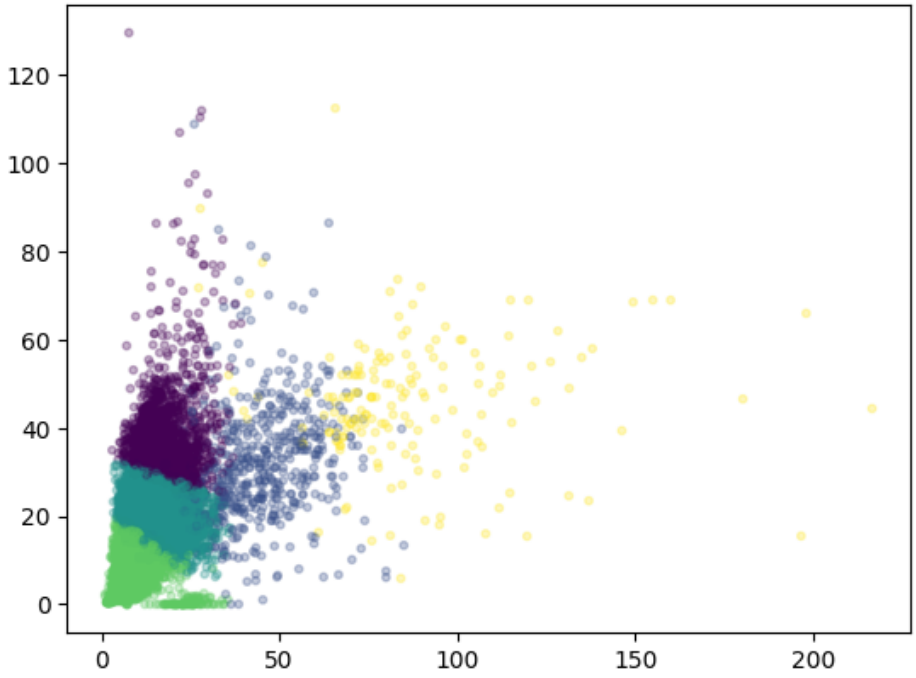
Random tests on other values of K:

In [65…

```
k = 5
kmeans = KMeans(n_clusters=k, init="random", n_init=1)
kmeans.fit(df_q4[["pm25_standardized", "pm10_standardized", "no2_standardized"]])
```

```
df_q4["label_test"] = pd.Series(kmeans.labels_, index=df_q4.index)

print(homogeneity_score(df_q4["development"], df_q4["label_test"]))
plt.scatter(
    x=df_q4["pm25_concentration"],
    y=df_q4["no2_concentration"],
    c=df_q4["label_test"],
    s=10,
    alpha=0.3,
)
plt.show()
```

```
0.25890651611878013
```



Question 5

Tianle Yao

In [66…

```
features = ["who_region", "year", "country_name", "no2_concentration"]  # features usec
df_q5 = df[(df["year"] >= 2010) & (df["year"] <= 2022)][
    features
]  # dataframe for question 5
df_q5.head()  # current df contains NaN values in no2_concentration column, clean it ir
```

Out[66…

	who_region	year	country_name	no2_concentration
0	4_Eur	2013.0	Spain	28.841
1	4_Eur	2014.0	Spain	19.575
2	4_Eur	2015.0	Spain	22.731
3	4_Eur	2016.0	Spain	20.204
4	4_Eur	2017.0	Spain	21.543

In [67…

```
df_q5 = df_q5.dropna(
    subset=["no2_concentration"]
)  # cleaned version of df_q5, drop rows that contains NaN in no2_concentration
df_q5.head()
```

Out[67…

	who_region	year	country_name	no2_concentration
0	4_Eur	2013.0	Spain	28.841
1	4_Eur	2014.0	Spain	19.575
2	4_Eur	2015.0	Spain	22.731
3	4_Eur	2016.0	Spain	20.204
4	4_Eur	2017.0	Spain	21.543

In [68…

```
df_q5["who_region"].value_counts()
```

Out[68…

```
who_region
4_Eur      22638
2_Amr       2812
6_Wpr        781
3_Sear       412
5_Emr        165
1_Afr        112
7_NonMS       14
Name: count, dtype: int64
```

In [69…

```
(df_q5[df_q5["who_region"] == "7_NonMS"]["country_name"]).value_counts()
# since these two countries show infrequently in this 13 year period, we will exclude t
```

Out[69…

```
country_name
occupied Palestinian territory, including east Jerusalem    13
Liechtenstein                                                1
Name: count, dtype: int64
```

In [70…

```
df_q5 = df_q5[
    (df_q5["country_name"] != "Liechtenstein")
    & (
        df_q5["country_name"]
        != "occupied Palestinian territory, including east Jerusalem"
    )
]
df_q5["who_region"].value_counts()
```

Out[70…

```
who_region
4_Eur     22638
2_Amr      2812
6_Wpr       781
3_Sear      412
5_Emr       165
1_Afr       112
Name: count, dtype: int64
```

- Since we want to predict the world level no2_concentration by 3 levels: each entry has the same influence; each country has a different influence according to its number of occurrences in the dataset; each region has a different influence according to its number of occurrences in the dataset

- The influence of an entry can be of the following 3 types: same weight; different weight according to their respective country; different weight according to their respective region in WHO

In [71…

```
# calculating weights for region and country levels

# use the number of occurence a country has in the dataframe to get the weight, more it
# create a dictionary so can create a new column in which the weight is mapped by the c
weight_country = pd.DataFrame(df_q5["country_name"].value_counts(normalize=True))[
    "proportion"
].to_dict()
df_q5.loc[:, "country_weight"] = df_q5["country_name"].map(weight_country)

# same as country level
weight_region = pd.DataFrame(df_q5["who_region"].value_counts(normalize=True))[
```

```
    "proportion"
].to_dict()
df_q5.loc[:, "region_weight"] = df_q5["who_region"].map(weight_region)

df_q5.head()
```

Out[71…

	who_region	year	country_name	no2_concentration	country_weight	region_weight
0	4_Eur	2013.0	Spain	28.841	0.113782	0.840936
1	4_Eur	2014.0	Spain	19.575	0.113782	0.840936
2	4_Eur	2015.0	Spain	22.731	0.113782	0.840936
3	4_Eur	2016.0	Spain	20.204	0.113782	0.840936
4	4_Eur	2017.0	Spain	21.543	0.113782	0.840936

In [72…

```
model = LinearRegression()
X_train = df_q5[["year"]]
y_train = df_q5[["no2_concentration"]]
X_test = pd.DataFrame([i for i in range(2023, 2036, 1)], columns=["year"])

df_q5.sort_values("no2_concentration", ascending=False)
```

Out[72…

	who_region	year	country_name	no2_concentration	country_weight	region_weight
39606	4_Eur	2019.0	Bosnia and Herzegovina	3670.314	0.002489	0.840936
30227	5_Emr	2020.0	Iraq	928.639	0.000483	0.006129
1628	5_Emr	2012.0	Iran (Islamic Republic of)	210.675	0.001337	0.006129
38968	5_Emr	2013.0	Iran (Islamic Republic of)	178.753	0.001337	0.006129
35740	4_Eur	2019.0	T√°rkiye	174.454	0.023923	0.840936
...	...	...	...	...	...	...
25719	6_Wpr	2019.0	Malaysia	0.004	0.000594	0.029012
29584	6_Wpr	2019.0	Malaysia	0.004	0.000594	0.029012
34678	6_Wpr	2019.0	Malaysia	0.003	0.000594	0.029012
26320	6_Wpr	2019.0	Malaysia	0.003	0.000594	0.029012
37503	6_Wpr	2019.0	Malaysia	0.002	0.000594	0.029012

26920 rows × 6 columns

In [73…

```
# each entry with equal weights
pipeline = make_pipeline(StandardScaler(), model)
scores = cross_val_score(
    pipeline, scoring="neg_mean_squared_error", X=X_train, y=y_train, cv=5
)
mse = (-scores).mean()
rmse = np.sqrt(mse)

mse, rmse
```

Out[73…

```
(664.2410760621624, 25.77287481175048)
```

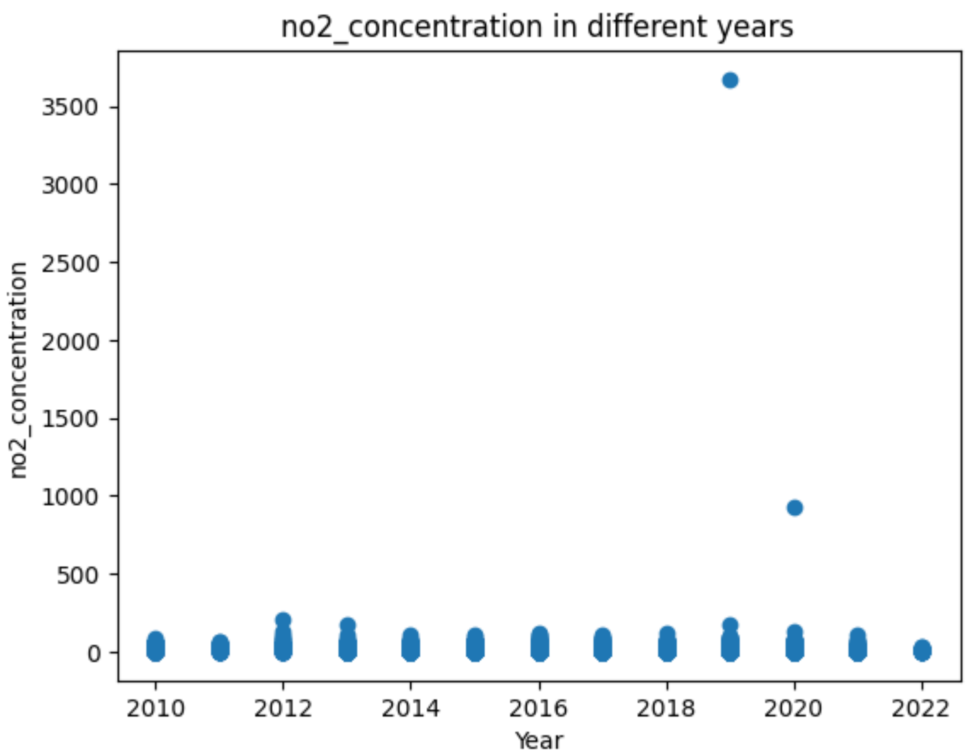
In [74…

```
pipeline.fit(X_train, y_train)
predictions = pipeline.predict(X_test)
predictions
```

Out[74… array([[14.54122785],
       [13.89343767],
       [13.24564749],
       [12.59785731],
       [11.95006713],
       [11.30227695],
       [10.65448677],
       [10.00669659],
       [9.35890641],
       [8.71111623],
       [8.06332605],
       [7.41553587],
       [6.76774569]])

In [75…
```
# Since each entry of the data is corresponding one city's no2_concentration
# the graph plotted below is actually all cities no2_concentration rather than the worl

plt.scatter(X_train, y_train)
plt.xlabel("Year")
plt.ylabel("no2_concentration")
plt.title("no2_concentration in different years")
```

Out[75… Text(0.5, 1.0, 'no2_concentration in different years')



In [76…
```
pipeline = make_pipeline(StandardScaler(), model)  # by country level

cv_results = cross_validate(
    pipeline,
    X=X_train,
    y=y_train,
    params={"linearregression__sample_weight": df_q5["country_weight"]},
    scoring="neg_mean_squared_error",
)
mse = (-cv_results["test_score"]).mean()
rmse = np.sqrt(mse)

mse, rmse
```

Out[76… (664.4680424037426, 25.7772776375579)

In [77…
```
# sample_weight of linearregression allows us to give more importance to certain sample
pipeline.fit(X_train, y_train, linearregression__sample_weight=df_q5["country_weight"])
predictions2 = pipeline.predict(X_test)
```

```
predictions2
```

Out[77…

```
array([[13.28276408],
       [12.48070053],
       [11.67863698],
       [10.87657342],
       [10.07450987],
       [ 9.27244632],
       [ 8.47038276],
       [ 7.66831921],
       [ 6.86625566],
       [ 6.0641921 ],
       [ 5.26212855],
       [ 4.460065  ],
       [ 3.65800144]])
```

In [78…

```
pipeline = make_pipeline(StandardScaler(), model)  # by region level

cv_results = cross_validate(
    pipeline,
    X=X_train,
    y=y_train,
    params={"linearregression__sample_weight": df_q5["region_weight"]},
    scoring="neg_mean_squared_error",
)
mse = (-cv_results["test_score"]).mean()
rmse = np.sqrt(mse)

mse, rmse
```

Out[78…

```
(664.2992104782239, 25.774002608796017)
```

In [79…

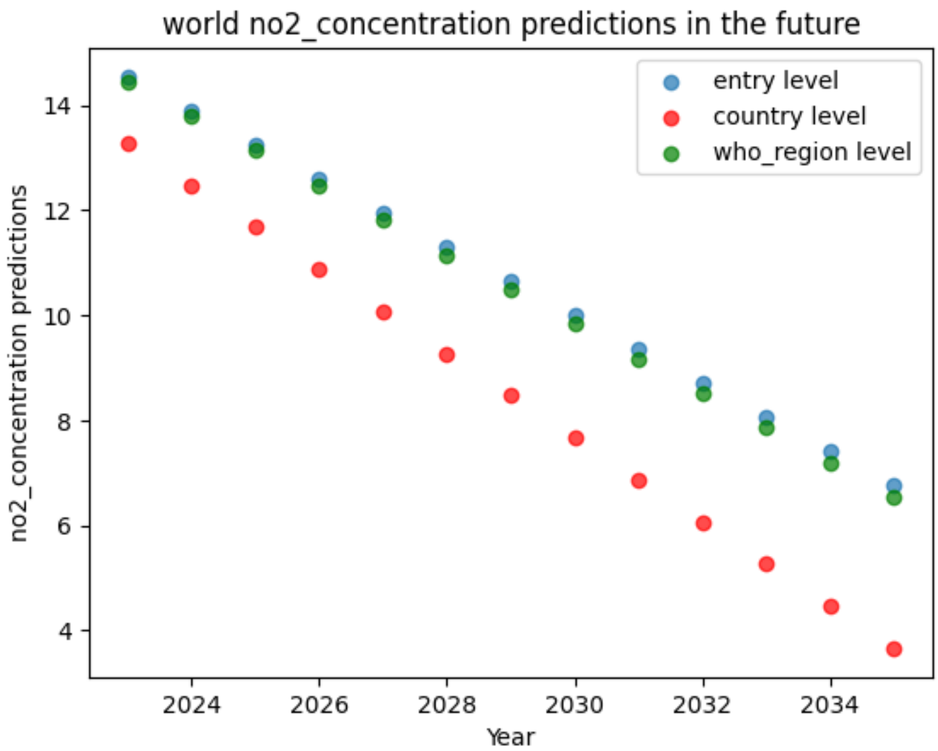
```
pipeline.fit(X_train, y_train, linearregression__sample_weight=df_q5["region_weight"])
predictions3 = pipeline.predict(X_test)
predictions3
```

Out[79…

```
array([[14.45040027],
       [13.79074919],
       [13.13109812],
       [12.47144705],
       [11.81179598],
       [11.15214491],
       [10.49249384],
       [ 9.83284277],
       [ 9.1731917 ],
       [ 8.51354063],
       [ 7.85388956],
       [ 7.19423849],
       [ 6.53458742]])
```

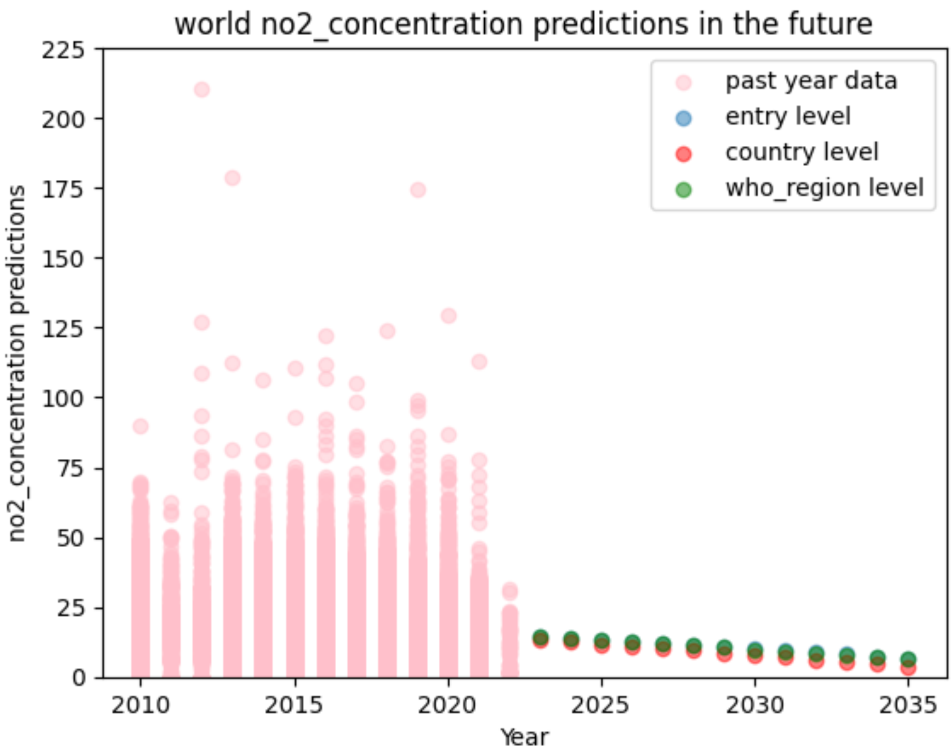
In [80…

```
fig01 = plt.scatter(
    X_test, predictions, alpha=0.7, label="entry level"
)  # each entry with same wights
fig02 = plt.scatter(
    X_test, predictions2, c="red", alpha=0.7, label="country level"
)  # each entry weighted by country
fig03 = plt.scatter(
    X_test, predictions3, c="green", alpha=0.7, label="who_region level"
)  # each entry weighted by WHO region

plt.xlabel("Year")
plt.ylabel("no2_concentration predictions")
plt.title("world no2_concentration predictions in the future")

plt.legend()
plt.savefig("5.1.png", dpi=500)
```



In [81…

```
fig00 = plt.scatter(
    X_train, y_train, c="pink", alpha=0.5, label="past year data"
)  # past year training data
fig01 = plt.scatter(
    X_test, predictions, alpha=0.5, label="entry level"
)  # each entry with same wights
fig02 = plt.scatter(
    X_test, predictions2, c="red", alpha=0.5, label="country level"
)  # each entry weighted by country
fig03 = plt.scatter(
    X_test, predictions3, c="green", alpha=0.5, label="who_region level"
)  # each entry weighted by WHO region

plt.xlabel("Year")
plt.ylabel("no2_concentration predictions")
plt.title("world no2_concentration predictions in the future")

plt.legend()
plt.ylim(0, 225)
plt.savefig("5.2.png", dpi=500)
```



Data Interpretation

- The predictions are much smaller than the past training data seen on the graph because they are predictions for the world no2_concentration as a whole, whereas the training data are no2_concentration specific to each city, so many 'outliers' have been averaged down a lot