


Upload the Dataset

```
from google.colab import files
uploaded = files.upload()
```



Choose Files

 No file chosen


Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving PRSA_data_2010.1.1-2014.12.31.csv to PRSA_data_2010.1.1-2014.12.31.csv

Load the Dataset

```
import pandas as pd

# Replace the file name if different
df = pd.read_csv('/content/PRSA_data_2010.1.1-2014.12.31.csv')
df.head()
```



	No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	Iws	Is	Ir
0	1	2010	1	1	0	NaN	-21	-11.0	1021.0	NW	1.79	0	0
1	2	2010	1	1	1	NaN	-21	-12.0	1020.0	NW	4.92	0	0
2	3	2010	1	1	2	NaN	-21	-11.0	1019.0	NW	6.71	0	0
3	4	2010	1	1	3	NaN	-21	-14.0	1019.0	NW	9.84	0	0
4	5	2010	1	1	4	NaN	-20	-12.0	1018.0	NW	12.97	0	0

Data Exploration

```
# Shape and info
print("Shape:", df.shape)
print("\nInfo:\n")
df.info()

# Describe
df.describe()
```

↗ Shape: (43824, 13)

Info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43824 entries, 0 to 43823
Data columns (total 13 columns):
#   Column      Non-Null Count  Dtype
---  -
0    No          43824 non-null  int64
1    year        43824 non-null  int64
2    month       43824 non-null  int64
3    day         43824 non-null  int64
4    hour        43824 non-null  int64
5    pm2.5       41757 non-null  float64
6    DEWP        43824 non-null  int64
7    TEMP        43824 non-null  float64
8    PRES        43824 non-null  float64
9    cbwd        43824 non-null  object
10   Iws         43824 non-null  float64
11   Is          43824 non-null  int64
12   Ir          43824 non-null  int64
dtypes: float64(4), int64(8), object(1)
memory usage: 4.3+ MB
```

	No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	
count	43824.000000	43824.000000	43824.000000	43824.000000	43824.000000	41757.000000	43824.000000	43824.000000	43824.000000	43824.0
mean	21912.500000	2012.000000	6.523549	15.727820	11.500000	98.613215	1.817246	12.448521	1016.447654	23.8
std	12651.043435	1.413842	3.448572	8.799425	6.922266	92.050387	14.433440	12.198613	10.268698	50.0
min	1.000000	2010.000000	1.000000	1.000000	0.000000	0.000000	-40.000000	-19.000000	991.000000	0.4
25%	10956.750000	2011.000000	4.000000	8.000000	5.750000	29.000000	-10.000000	2.000000	1008.000000	1.7
50%	21912.500000	2012.000000	7.000000	16.000000	11.500000	72.000000	2.000000	14.000000	1016.000000	5.3
75%	32868.250000	2013.000000	10.000000	23.000000	17.250000	137.000000	15.000000	23.000000	1025.000000	21.9
max	43824.000000	2014.000000	12.000000	31.000000	23.000000	994.000000	28.000000	42.000000	1046.000000	585.6

Check for Missing Values and Duplicates

```
# Missing values
print("Missing values:\n", df.isnull().sum())

# Duplicates
print("\nDuplicate rows:", df.duplicated().sum())

# Drop duplicates if necessary
df = df.drop_duplicates()
```

↗ Missing values:

No	0
year	0
month	0
day	0
hour	0
pm2.5	2067
DEWP	0
TEMP	0
PRES	0
cbwd	0
Iws	0
Is	0
Ir	0

dtype: int64

Duplicate rows: 0

Visualize a Few Features

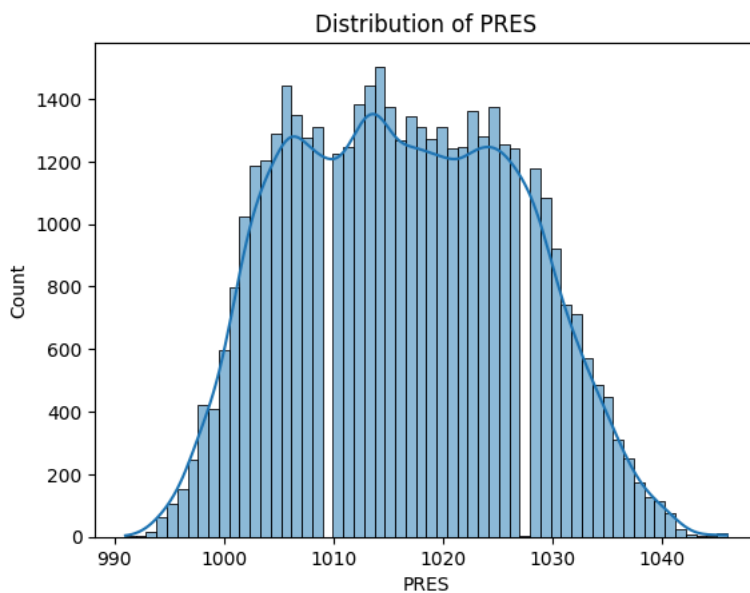
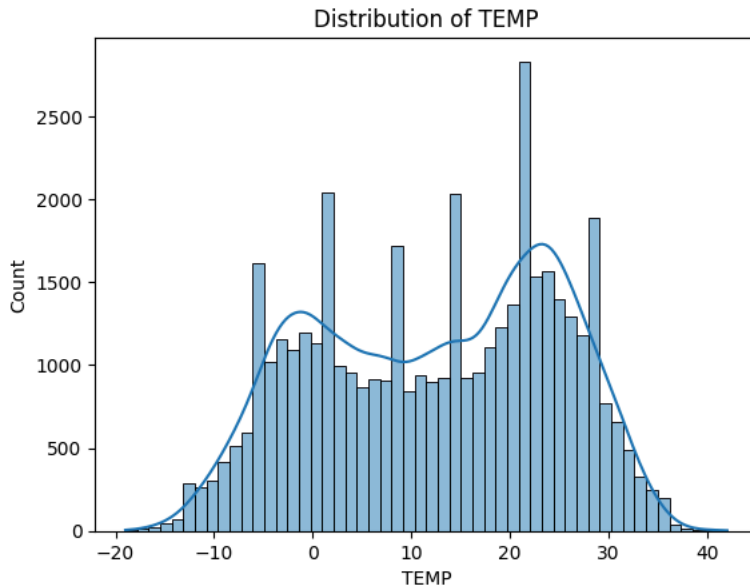
```
import matplotlib.pyplot as plt
import seaborn as sns

# Distribution of a few numeric features
```

```

features = ['TEMP', 'PRES', 'DEWP', 'pm2.5']
for feature in features:
    sns.histplot(df[feature].dropna(), kde=True)
    plt.title(f'Distribution of {feature}')
    plt.show()

```



```

# Assuming 'pm2.5' is the target variable
target = 'pm2.5'
features = df.drop(columns=[target]).columns.tolist()
print("Features:", features)

```



```

features = ['No', 'year', 'month', 'day', 'hour', 'DEWP', 'TEMP', 'PRES', 'cbwd', 'Iws', 'Is', 'Ir']

```

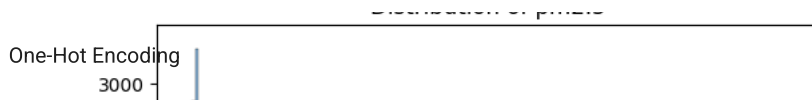
Convert Categorical Columns to Numerical

```

# Example: converting 'cbwd' (wind direction) to numerical codes
if 'cbwd' in df.columns:

```

```
df['cbwd'] = df['cbwd'].astype('category').cat.codes
```

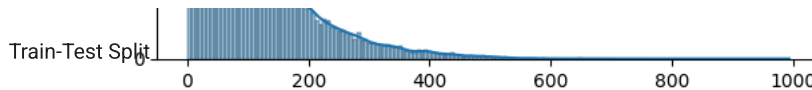


```
df = pd.get_dummies(df, columns=df.select_dtypes(include='object').columns)
```



```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.drop(columns=[target]))
X = pd.DataFrame(scaled_features, columns=df.drop(columns=[target]).columns)
y = df[target]
```



```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Building

```
# Drop rows where target (pm2.5) is NaN
df = df.dropna(subset=['pm2.5'])
```

```
# Now reassign X and y
X = df.drop(columns=['pm2.5'])
y = df['pm2.5']
```

```
# Also, re-encode and scale as before
X['cbwd'] = X['cbwd'].astype('category').cat.codes
X = pd.get_dummies(X, drop_first=True)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
model = RandomForestRegressor()
model.fit(X_train, y_train)
```

```
RandomForestRegressor
```

Evaluation

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
y_pred = model.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))
```


```
MSE: 1212.5156050766286
R^2 Score: 0.8623941962718467
```

Make Predictions from New Input

```
import numpy as np

# If X_test is a DataFrame
if hasattr(X_test, 'iloc'):
    sample = X_test.iloc[0].values.reshape(1, -1)
else: # If it's a NumPy array
    sample = X_test[0].reshape(1, -1)

prediction = model.predict(sample)
print("Predicted PM2.5:", prediction[0])
```


 Predicted PM2.5: 142.03

Convert to DataFrame and Encode

```
# Simulate user input
user_input = {'TEMP': 5, 'PRES': 1020, 'DEWP': -3, 'cbwd': 'NW', 'Iws': 10, 'Is': 0, 'Ir': 0}
user_df = pd.DataFrame([user_input])

# Convert categorical columns
if 'cbwd' in user_df.columns:
    user_df['cbwd'] = user_df['cbwd'].astype('category').cat.codes

print("Scaler was fitted on:", scaler.feature_names_in_)
print("User input columns: ", user_df.columns.tolist())
```

 Scaler was fitted on: ['No' 'year' 'month' 'day' 'hour' 'DEWP' 'TEMP' 'PRES' 'cbwd' 'Iws' 'Is' 'Ir']
User input columns: ['TEMP', 'PRES', 'DEWP', 'cbwd', 'Iws', 'Is', 'Ir']

Predict the Final Grade (PM2.5)

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Simulated training data (replace with your actual data)
data = {
    'TEMP': [22, 25, 20, 18, 23, 19],
    'PRES': [1015, 1020, 1012, 1018, 1016, 1022],
    'DEWP': [10, 12, 8, 7, 9, 11],
    'cbwd': ['NW', 'NE', 'SE', 'SW', 'NW', 'SE'],
    'Iws': [15, 18, 13, 12, 14, 16],
    'Is': [5, 3, 4, 6, 5, 4],
    'Ir': [0, 1, 0, 0, 1, 0],
    'PM2.5': [25, 30, 20, 15, 28, 22] # Target variable (Final Grade - PM2.5)
}

df = pd.DataFrame(data)

# Preprocessing
# Handle categorical variable 'cbwd' (convert to numerical codes)
df['cbwd'] = df['cbwd'].astype('category').cat.codes

# Feature columns and target variable
X = df.drop('PM2.5', axis=1)
y = df['PM2.5']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Model training (Random Forest Regressor)
model = RandomForestRegressor(random_state=42)
model.fit(X_train_scaled, y_train)

# Prediction for new data (simulated user input)
user_input = {'TEMP': 5, 'PRES': 1020, 'DEWP': -3, 'cbwd': 'NW', 'Iws': 10, 'Is': 0, 'Ir': 0}
user_df = pd.DataFrame([user_input])

# Convert categorical feature (same as training time)
user_df['cbwd'] = user_df['cbwd'].astype('category').cat.codes

# Scale the user input
user_df_scaled = scaler.transform(user_df)

# Make prediction
final_prediction = model.predict(user_df_scaled)
print("Final Predicted PM2.5:", final_prediction[0])
```

Final Predicted PM2.5: 19.66

Deployment - Building an Interactive App

```
!pip install gradio
import gradio as gr
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.31.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.4)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.11.9-py3-none-manylinux_2_17_x86_64_manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (13.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.16)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
```

Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
 Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1)
 Requirement already satisfied: hf-xet<2.0.0,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio)
 Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9.
 Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
 Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
 Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0

Create a Prediction Function

```
def predict_pm25(year, month, day, hour, DEWP, TEMP, PRES, Iws, cbwd):
    import numpy as np
    import pandas as pd

    # Ensure cbwd matches the one-hot or categorical format you used
    cbwd_map = {'NW': 0, 'NE': 1, 'SE': 2, 'SW': 3}
    cbwd_num = cbwd_map.get(cbwd, 0) # default to 0 if not found

    # Construct the input using the same order and columns
    input_data = pd.DataFrame([[year, month, day, hour, DEWP, TEMP, PRES, Iws, cbwd_num]],
                              columns=['year', 'month', 'day', 'hour', 'DEWP', 'TEMP', 'PRES', 'Iws', 'cbwd'])

    # Convert cbwd to same encoding as during training
    input_data['cbwd'] = input_data['cbwd'].astype('int')

    # Apply the same feature scaling
    input_scaled = scaler.transform(input_data)

    # Predict
    prediction = model.predict(input_scaled)
    return f"Predicted PM2.5: {prediction[0]:.2f}"
```

Create the Gradio Interface

```
def predict_pm25(year, month, day, hour, DEWP, TEMP, PRES, Iws, cbwd):
    import pandas as pd
    import numpy as np

    # Map wind direction to numeric (same as in training)
    cbwd_map = {'NW': 0, 'NE': 1, 'SE': 2, 'SW': 3}
    cbwd_num = cbwd_map.get(cbwd, 0) # Default to 0 if unknown

    # Create DataFrame (ensure same column names and order used in training)
    input_df = pd.DataFrame([[year, month, day, hour, DEWP, TEMP, PRES, Iws, cbwd_num]],
                             columns=['year', 'month', 'day', 'hour', 'DEWP', 'TEMP', 'PRES', 'Iws', 'cbwd'])

    # Scale features
    try:
        input_scaled = scaler.transform(input_df)
    except Exception as e:
        return f"Scaling Error: {str(e)}"

    # Predict
    try:
        prediction = model.predict(input_scaled)
        return f"Predicted PM2.5: {prediction[0]:.2f}"
    except Exception as e:
        return f"Prediction Error: {str(e)}"

import gradio as gr

iface = gr.Interface(
    fn=predict_pm25,
    inputs=[
        gr.Number(label="Year"),
        gr.Number(label="Month"),
        gr.Number(label="Day"),
        gr.Number(label="Hour"),
        gr.Number(label="Dew Point (DEWP)"),
        gr.Number(label="Temperature (TEMP)"),
        gr.Number(label="Pressure (PRES)"),
        gr.Number(label="Wind Speed (Iws)"),
    ],
    outputs=[
        gr.Number(label="Predicted PM2.5")
    ],
    title="PM2.5 Prediction Interface",
    description="Enter weather data to predict PM2.5 concentration."
)
```

```
gr.Dropdown(["NW", "NE", "SE", "SW"], label="Wind Direction (cbwd)")
],
outputs="text",
title="Air Quality PM2.5 Predictor",
description="Enter weather values to predict air pollution levels."
)

iface.launch()
```

↗ It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: <https://d3a344731773db0a56.gradio.live>
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working dir

Air Quality PM2.5 Predictor

Enter weather values to predict air pollution levels.

Year

0

Month

0

output

Flag