

Practical ML & DE

Project 1
&
Project 2

Abdulaziz Saleh Alhasil

Project 1

```
In [1]: import numpy as np

import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import LabelEncoder
```

```
In [2]: warnings.filterwarnings("ignore")

# Load dataset
df = pd.read_csv('Sleep_health_and_lifestyle_dataset.csv')
```

```
In [ ]: # Data Exploration
print(df.head())
print(df.dtypes)
print(df.columns)
print(df.isnull().sum())
```

```
In [17]: df.describe()
```

```
Out[17]:
```

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level
count	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000
mean	187.500000	0.505348	42.184492	3.772727	7.132086	7.312834	59.171123	5.385027
std	108.108742	0.500641	8.673133	3.056081	0.795657	1.196956	20.830804	1.774526
min	1.000000	0.000000	27.000000	0.000000	5.800000	4.000000	30.000000	3.000000
25%	94.250000	0.000000	35.250000	1.000000	6.400000	6.000000	45.000000	4.000000
50%	187.500000	1.000000	43.000000	3.000000	7.200000	7.000000	60.000000	5.000000
75%	280.750000	1.000000	50.000000	5.000000	7.800000	8.000000	75.000000	7.000000
max	374.000000	1.000000	59.000000	10.000000	8.500000	9.000000	90.000000	8.000000

```
In [4]: # Drop columns with more than 30% missing values
missing_percentage = df.isnull().mean() * 100
columns_to_drop = missing_percentage[missing_percentage > 30].index
df = df.drop(columns=columns_to_drop)
```

```
In [5]: print(columns_to_drop)
```

```
Index(['Sleep Disorder'], dtype='str')
```

```
In [6]: print(df.columns)
```

```
Index(['Person ID', 'Gender', 'Age', 'Occupation', 'Sleep Duration',  
      'Quality of Sleep', 'Physical Activity Level', 'Stress Level',  
      'BMI Category', 'Blood Pressure', 'Heart Rate', 'Daily Steps'],  
      dtype='str')
```

```
In [7]: # Label Encoding for categorical columns  
label_encoder = LabelEncoder()  
for column in df.select_dtypes(include=['object']).columns:  
    df[column] = label_encoder.fit_transform(df[column].astype(str))
```

```
In [8]: df.head()
```

```
Out[8]:
```

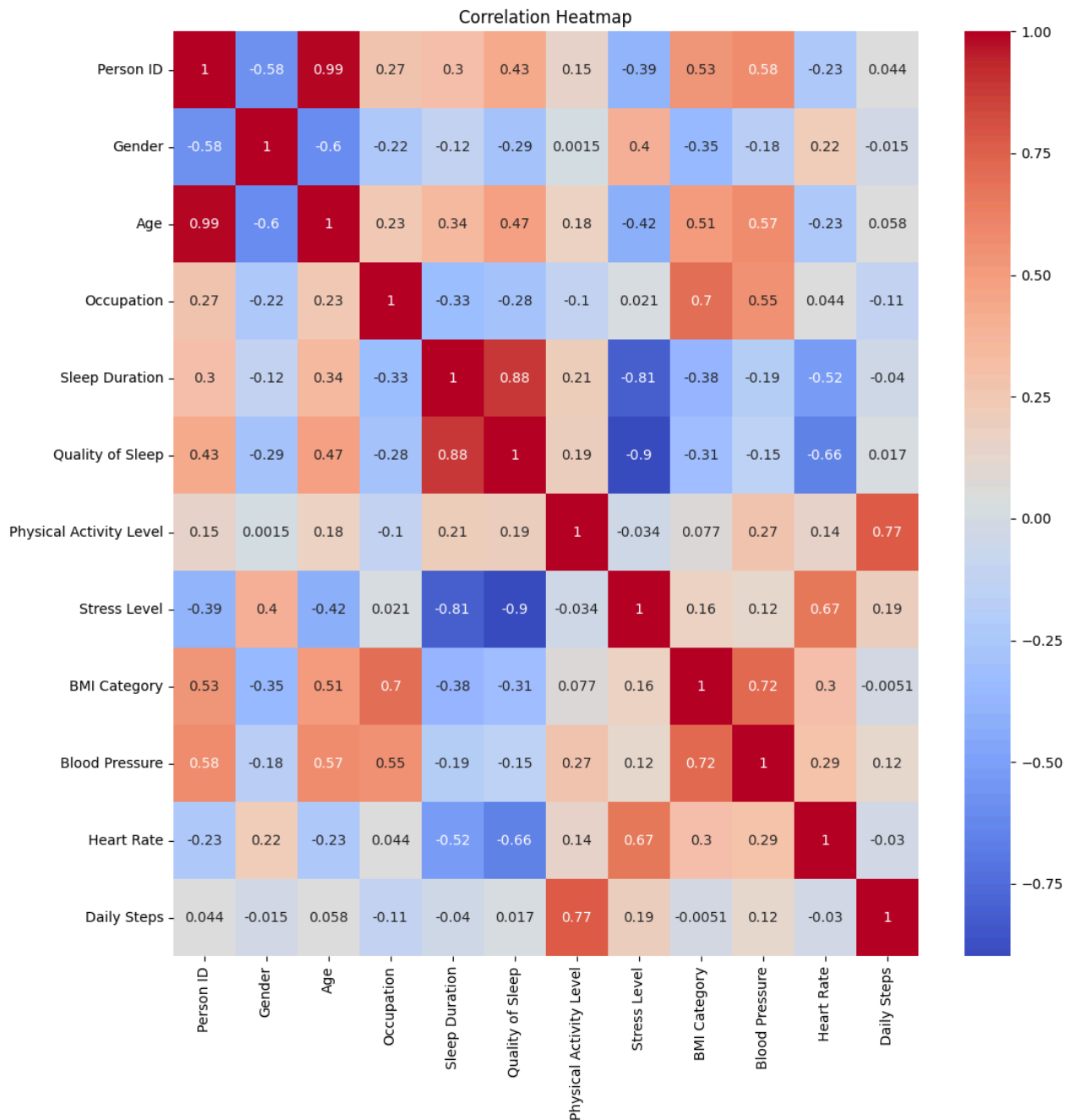
	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate	D St
0	1	1	27	9	6.1	6	42	6	3	11	77	4
1	2	1	28	1	6.2	6	60	8	0	9	75	10
2	3	1	28	1	6.2	6	60	8	0	9	75	10
3	4	1	28	6	5.9	4	30	8	2	22	85	3
4	5	1	28	6	5.9	4	30	8	2	22	85	3



```
In [9]: # Fill missing values for numerical columns (Mean)  
for column in df.select_dtypes(include=['float64', 'int64']).columns:  
    df[column] = df[column].fillna(df[column].mean())
```

```
In [11]: # Fill missing values for categorical columns (Mode)  
for column in df.select_dtypes(include=['object']).columns:  
    df[column] = df[column].fillna(df[column].mode()[0])
```

```
In [12]: # Visualizations  
plt.figure(figsize=(12, 12))  
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')  
plt.title('Correlation Heatmap')  
plt.show()
```



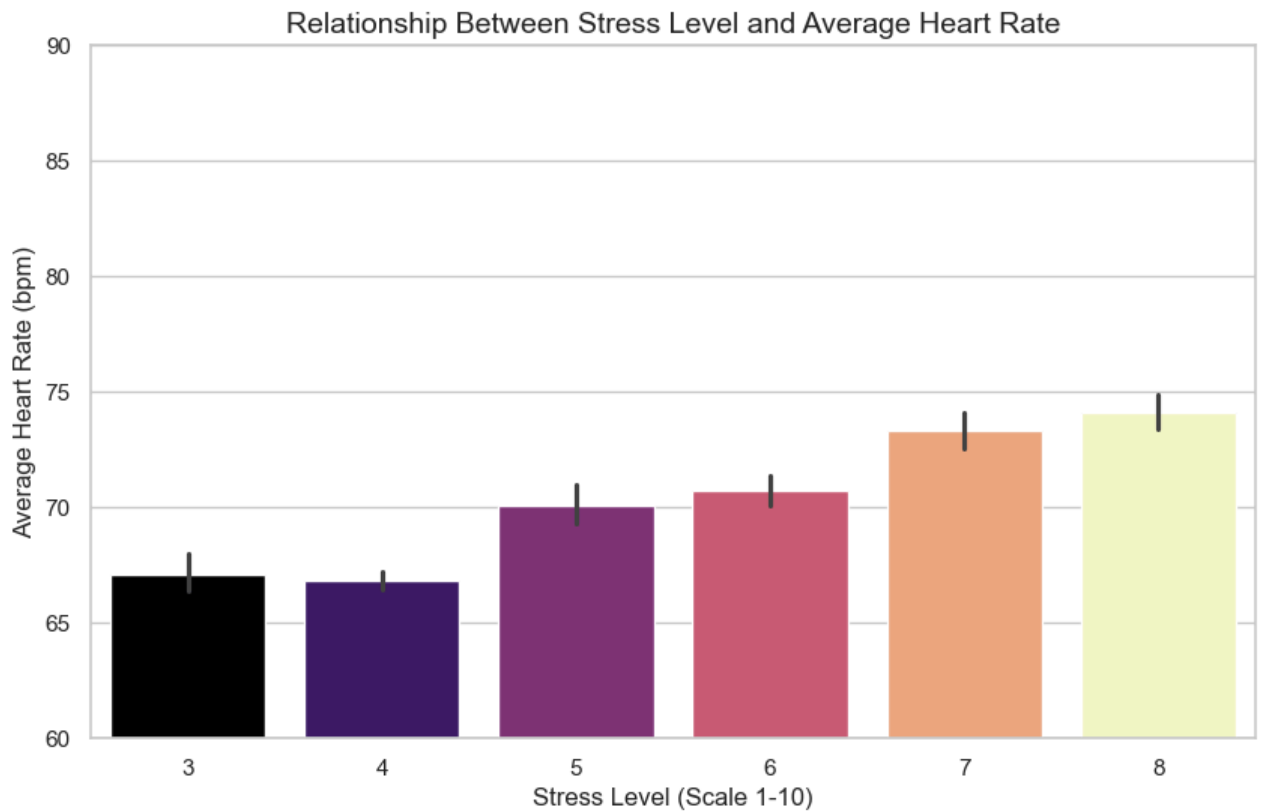
```
In [16]: # 2. Set the visual style
plt.figure(figsize=(10, 6))
sns.set_theme(style="whitegrid")

# 3. Create the Bar Chart
# sns.barplot automatically calculates the mean for each category
ax = sns.barplot(x='Stress Level', y='Heart Rate', data=df, palette='magma', hue='Stres

# 4. Adding Labels and title
plt.title('Relationship Between Stress Level and Average Heart Rate', fontsize=14)
plt.xlabel('Stress Level (Scale 1-10)', fontsize=12)
plt.ylabel('Average Heart Rate (bpm)', fontsize=12)

# 5. Adjust y-axis to see the differences more clearly
plt.ylim(60, 90)
```

```
# Display the plot
plt.show()
```



In [25]:

```
# 2. Extract Systolic Blood Pressure (the first number)
df['Systolic BP'] = df['Blood Pressure'].apply(lambda x: int(x.split('/')[0]))

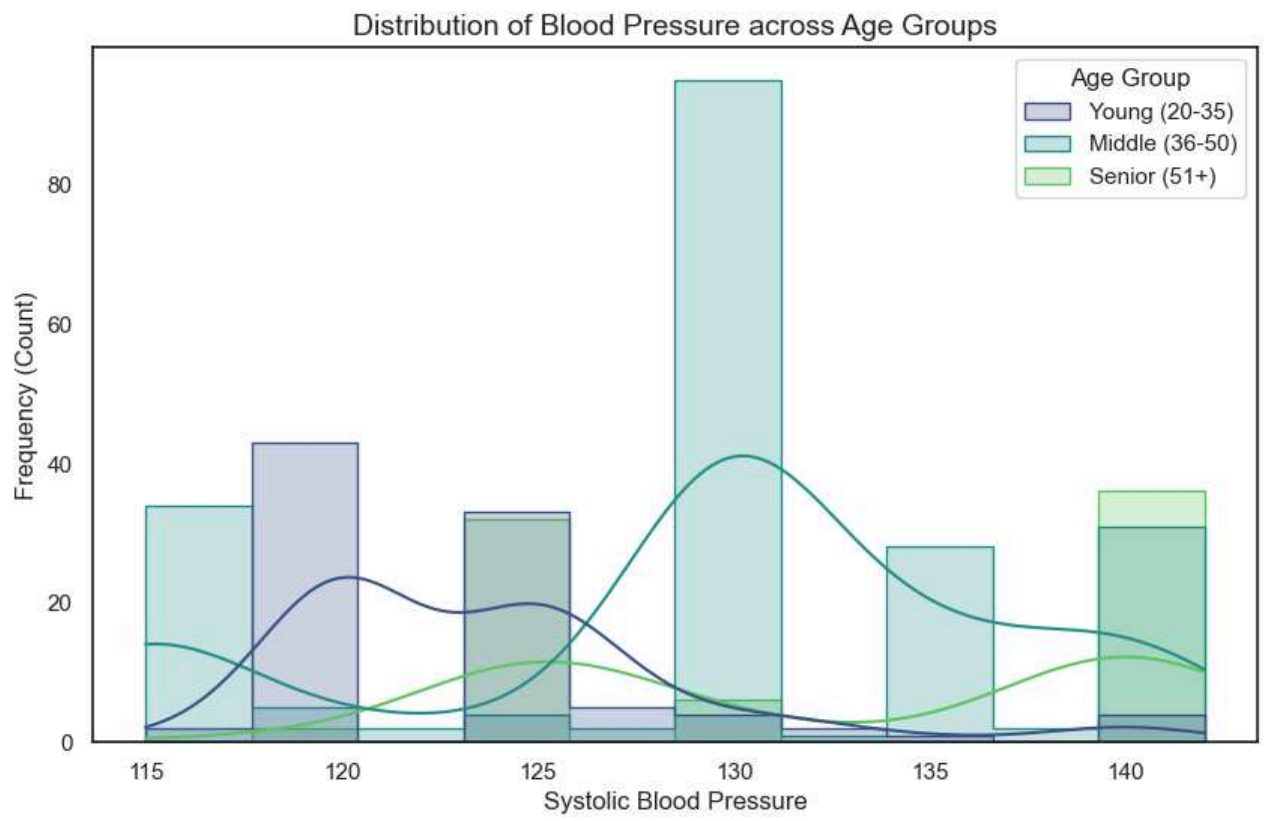
# 3. Categorize Age into groups for a clearer histogram view
df['Age Group'] = pd.cut(df['Age'], bins=[20, 35, 50, 65], labels=['Young (20-35)', 'Mi

# 4. Set plot style
plt.figure(figsize=(10, 6))

# 5. Draw the Histogram with multiple layers (one for each age group)
sns.histplot(data=df, x='Systolic BP', hue='Age Group', kde=True, element="step", palet

# 6. Add Labels and title
plt.title('Distribution of Blood Pressure across Age Groups', fontsize=14)
plt.xlabel('Systolic Blood Pressure', fontsize=12)
plt.ylabel('Frequency (Count)', fontsize=12)

plt.show()
```



In []:

Project 2

- Step 1: Download and Load the Dataset

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset using pandas
df = pd.read_csv('Air_Quality.csv')

# Display the first 5 rows to understand the data structure
print("Dataset Head:")
print(df.head())

# Check for data types and non-null counts
print("\nDataset Info:")
print(df.info())
```

Dataset Head:

	Unique ID	Indicator ID	Name \
0	179772	640	Boiler Emissions- Total SO2 Emissions
1	179785	640	Boiler Emissions- Total SO2 Emissions
2	178540	365	Fine particles (PM 2.5)
3	178561	365	Fine particles (PM 2.5)
4	823217	365	Fine particles (PM 2.5)

	Measure	Measure Info	Geo Type	Name	Geo Join ID \
0	Number per km2	number	UHF42		409.0
1	Number per km2	number	UHF42		209.0
2	Mean	mcg/m3	UHF42		209.0
3	Mean	mcg/m3	UHF42		409.0
4	Mean	mcg/m3	UHF42		409.0

	Geo Place Name	Time Period	Start_Date	Data Value \
0	Southeast Queens	2015	01/01/2015	0.3
1	Bensonhurst - Bay Ridge	2015	01/01/2015	1.2
2	Bensonhurst - Bay Ridge	Annual Average 2012	12/01/2011	8.6
3	Southeast Queens	Annual Average 2012	12/01/2011	8.0
4	Southeast Queens	Summer 2022	06/01/2022	6.1

Message

0	NaN
1	NaN
2	NaN
3	NaN
4	NaN

Dataset Info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18025 entries, 0 to 18024
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Unique ID             18025 non-null  int64
```

```

1 Indicator ID      18025 non-null  int64
2 Name             18025 non-null  object
3 Measure          18025 non-null  object
4 Measure Info     18025 non-null  object
5 Geo Type Name    18025 non-null  object
6 Geo Join ID      18016 non-null  float64
7 Geo Place Name   18016 non-null  object
8 Time Period      18025 non-null  object
9 Start_Date       18025 non-null  object
10 Data Value      18025 non-null  float64
11 Message         0 non-null     float64
dtypes: float64(3), int64(2), object(7)
memory usage: 1.7+ MB
None

```

```
In [2]: print(df.describe())
```

```

count      Unique ID  Indicator ID  Geo Join ID  Data Value  Message
mean    426387.692705    434.830180  5.906303e+05    21.428616    NaN
std    250489.450630    115.852371  7.769549e+06    23.999345    NaN
min    121644.000000    365.000000  1.000000e+00    0.000000    NaN
25%    175303.000000    365.000000  2.020000e+02    8.900000    NaN
50%    410803.000000    375.000000  3.030000e+02    15.200000    NaN
75%    649893.000000    386.000000  4.040000e+02    26.700000    NaN
max    828353.000000    661.000000  1.051061e+08    424.700000    NaN
**

```

- Step 2: Handle Missing Values and Data Preprocessing

```
In [3]: # Check for missing values in each column
print("\nMissing values before cleaning:\n", df.isnull().sum())
```

```

Missing values before cleaning:
Unique ID      0
Indicator ID    0
Name           0
Measure        0
Measure Info    0
Geo Type Name   0
Geo Join ID     9
Geo Place Name  9
Time Period     0
Start_Date      0
Data Value      0
Message        18025
dtype: int64

```

```
In [4]: df
```

```

Out[4]:
   Unique ID  Indicator ID  Name  Measure  Measure Info  Geo Type Name  Geo Join ID  Geo Place Name  Time Period  Start_Date
0  179772      640  Boiler Emissions-  Number per km2  number  UHF42  409.0  Southeast Queens  2015  01/01/2015

```

	Unique ID	Indicator ID	Name	Measure	Measure Info	Geo Type Name	Geo Join ID	Geo Place Name	Time Period	Start_Date
			Total SO2 Emissions							
1	179785	640	Boiler Emissions- Total SO2 Emissions	Number per km2	number	UHF42	209.0	Bensonhurst - Bay Ridge	2015	01/01/2015
2	178540	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	209.0	Bensonhurst - Bay Ridge	Annual Average 2012	12/01/2011
3	178561	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	409.0	Southeast Queens	Annual Average 2012	12/01/2011
4	823217	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	409.0	Southeast Queens	Summer 2022	06/01/2022
...	...	...	...	...	...	...	...	...	...	...
18020	816914	643	Annual vehicle miles traveled	Million miles	per square mile	CD	503.0	Tottenville and Great Kills (CD3)	2019	01/01/2019
18021	816913	643	Annual vehicle miles traveled	Million miles	per square mile	CD	503.0	Tottenville and Great Kills (CD3)	2010	01/01/2010
18022	816872	643	Annual vehicle miles traveled	Million miles	per square mile	UHF42	208.0	Canarsie - Flatlands	2010	01/01/2010
18023	816832	643	Annual vehicle miles traveled	Million miles	per square mile	UHF42	407.0	Southwest Queens	2010	01/01/2010
18024	151658	643	Annual vehicle miles traveled	Million miles	per square mile	UHF42	408.0	Jamaica	2005	01/01/2005

18025 rows × 12 columns

In [5]:

```

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

# Drop columns that are irrelevant for the machine learning model
# 'Message' is mostly empty, 'Start_Date' is a string date
cols_to_drop = ['Message', 'Start_Date', 'Unique ID', 'Indicator ID']
df = df.drop(columns=cols_to_drop)

```

```

# Impute missing values in the target column 'Data Value' with the mean
imputer = SimpleImputer(strategy='mean')
df['Data Value'] = imputer.fit_transform(df[['Data Value']])

# Convert categorical text columns into numerical values using LabelEncoder
le = LabelEncoder()
categorical_columns = ['Name', 'Measure', 'Measure Info', 'Geo Type Name', 'Geo Place N

for col in categorical_columns:
    # Fill any potential NaNs in categorical columns with 'Unknown' before encoding
    df[col] = le.fit_transform(df[col].astype(str))

print("\nData cleaning and encoding completed.")

```

Data cleaning and encoding completed.

**

- Step 3: Apply Machine Learning with Grid Search Optimizer

In [10]:

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer

# --- Pre-processing Step (Important to fix NaN error) ---

# 1. Handle Categorical Columns (Convert text to numbers)
# Let's identify text columns and encode them
le = LabelEncoder()
for col in df.columns:
    if df[col].dtype == 'object': # If the column is text
        df[col] = le.fit_transform(df[col].astype(str))

# 2. Handle Missing Values in the entire dataframe
# Sometimes scaling fails if there are NaNs
imputer = SimpleImputer(strategy='mean')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

# --- Now proceed to Step 3 (Model Building) ---

# Define features (X) and target (y)
X = df_imputed.drop('Data Value', axis=1)
y = df_imputed['Data Value']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define parameters to test
param_grid = {
    'n_estimators': [50, 100],

```

```

    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5]
}

# Run Grid Search
# Note: I changed scoring to 'neg_mean_squared_error' to be more robust
grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid,
    cv=3,
    scoring='r2',
    n_jobs=-1,
    error_score='raise' # This will show the exact error if it fails again
)

try:
    grid_search.fit(X_train_scaled, y_train)
    print(f"Best Hyperparameters: {grid_search.best_params_}")
    print(f"Best R2 Score: {grid_search.best_score_:.4f}")
except Exception as e:
    print(f"Detailed Error: {e}")

```

Best Hyperparameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
 Best R2 Score: 0.9347

- Step 4: Use Two Feature Selection Methods

First method: Filter Method:

```

In [11]: from sklearn.feature_selection import SelectKBest, f_regression

# 1. Filter Method: SelectKBest
# We select the top 5 features based on statistical correlation (f_regression)
# This method evaluates each feature independently before the model training
filter_selector = SelectKBest(score_func=f_regression, k=5)
filter_selector.fit(X_train_scaled, y_train)

# Get the names of the selected columns
filter_features = X.columns[filter_selector.get_support()].tolist()

print(f"Top 5 features selected by Filter Method: {filter_features}")

```

Top 5 features selected by Filter Method: ['Name', 'Measure', 'Measure Info', 'Geo Type Name', 'Time Period']

Second method: Embedded Method:

```

In [12]: # 2. Embedded Method: Feature Importance
# This method extracts feature importance directly from the trained model (Random Forest)
# It learns which features are most useful during the training process
best_rf_model = grid_search.best_estimator_ # Using the best model from Step 3
importances = best_rf_model.feature_importances_

# Sort and get the names of the top 5 features
indices = np.argsort(importances)[-5:]

```

```

embedded_features = [X.columns[i] for i in indices]

print(f"Top 5 features selected by Embedded Method: {embedded_features}")

```

Top 5 features selected by Embedded Method: ['Measure', 'Geo Place Name', 'Measure Info', 'Geo Join ID', 'Name']

• Step 5: Train Models on Selected Features and Evaluate

At this Step, we will use the features we selected using the Embedded Method (because it is usually more accurate) to retrain and evaluate the model.

```

In [13]: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# 1. Select only the important features (from Step 4)
# We will use 'embedded_features' which we identified earlier
X_train_final = X_train[embedded_features]
X_test_final = X_test[embedded_features]

# 2. Re-scaling the selected features
# Scale the final features to ensure consistent data range
scaler_final = StandardScaler()
X_train_final_scaled = scaler_final.fit_transform(X_train_final)
X_test_final_scaled = scaler_final.transform(X_test_final)

# 3. Initialize the Final Model
# Use the best hyperparameters found by Grid Search in Step 3
final_model = RandomForestRegressor(**grid_search.best_params_, random_state=42)

# 4. Train the final model on the selected features
final_model.fit(X_train_final_scaled, y_train)

# 5. Make final predictions on the test set
y_pred_final = final_model.predict(X_test_final_scaled)

# 6. Evaluation Metrics (As per Lecture 06)
# MAE: Average absolute error between actual and predicted values
# RMSE: Square root of the average squared errors (penalizes large errors)
# R2 Score: Percentage of variance explained by the model
mae = mean_absolute_error(y_test, y_pred_final)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_final))
r2 = r2_score(y_test, y_pred_final)

print("\n--- Final Project Evaluation Results ---")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"R-Squared (R2) Score: {r2:.4f}")

# 7. Visualization: Actual vs Predicted
# This plot helps us see how close our predictions are to the real values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_final, alpha=0.5, color='darkcyan')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2) # Diagonal line
plt.xlabel('Actual Air Quality Values')
plt.ylabel('Predicted Air Quality Values')
plt.title('Final Model Performance: Actual vs Predicted')
plt.show()

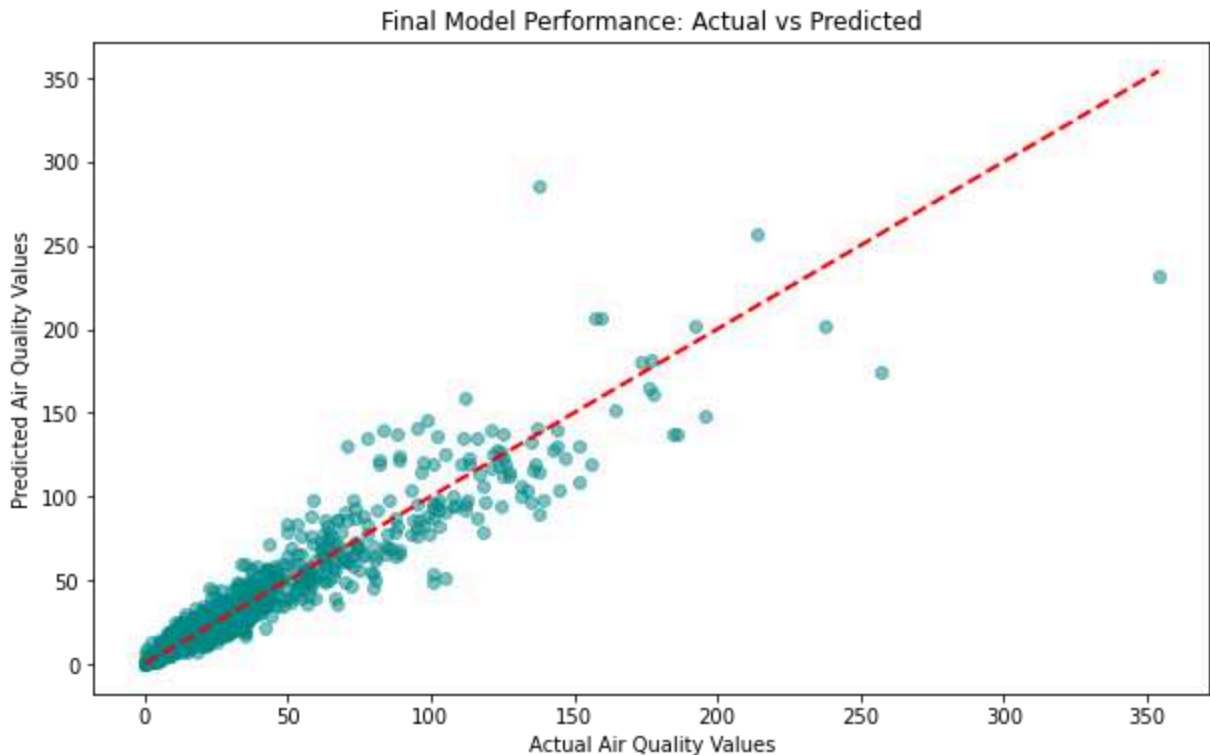
```

--- Final Project Evaluation Results ---

Mean Absolute Error (MAE): 3.9002

Root Mean Squared Error (RMSE): 7.5658

R-Squared (R2) Score: 0.8946



```
In [14]: # Expanded parameter grid for better accuracy
param_grid_optimized = {
    'n_estimators': [100, 200, 300], # Increased number of trees
    'max_depth': [None, 20, 30],      # Allowing deeper trees for IoT patterns
    'min_samples_split': [2, 5],
    'max_features': ['sqrt', 'log2']  # Adding feature variety to each tree
}

grid_search_opt = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid_optimized,
    cv=5, # Higher CV for more stable results
    scoring='r2',
    n_jobs=-1
)

grid_search_opt.fit(X_train_scaled, y_train)
print(f"New Best Parameters: {grid_search_opt.best_params_}")
```

New Best Parameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_split': 2, 'n_estimators': 200}

```
In [15]: # Increase k to 7 to capture more patterns
selector_opt = SelectKBest(score_func=f_regression, k=7)
selector_opt.fit(X_train, y_train)
selected_features_opt = X.columns[selector_opt.get_support()].tolist()

print(f"Optimized Selected Features: {selected_features_opt}")
```

Optimized Selected Features: ['Name', 'Measure', 'Measure Info', 'Geo Type Name', 'Geo Join ID', 'Geo Place Name', 'Time Period']

In [16]:

```
# Re-training with optimized settings
X_train_final = X_train[selected_features_opt]
X_test_final = X_test[selected_features_opt]

final_model_opt = grid_search_opt.best_estimator_
final_model_opt.fit(X_train_final, y_train)

y_pred_opt = final_model_opt.predict(X_test_final)

# New Results
print("\n--- Optimized Results ---")
print(f"MAE: {mean_absolute_error(y_test, y_pred_opt):.4f}")
print(f"R2 Score: {r2_score(y_test, y_pred_opt):.4f}")
```

```
--- Optimized Results ---
MAE: 2.4422
R2 Score: 0.9289
```

In [22]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Re-training with optimized settings
X_train_final = X_train[selected_features_opt]
X_test_final = X_test[selected_features_opt]

final_model_opt = grid_search_opt.best_estimator_
final_model_opt.fit(X_train_final, y_train)

y_pred_opt = final_model_opt.predict(X_test_final)

# Calculate RMSE
# RMSE is the square root of Mean Squared Error
rmse_val = np.sqrt(mean_squared_error(y_test, y_pred_opt))

# New Results
print("\n--- Optimized Results ---")
print(f"MAE: {mean_absolute_error(y_test, y_pred_opt):.4f}")
print(f"RMSE: {rmse_val:.4f}")
print(f"R2 Score: {r2_score(y_test, y_pred_opt):.4f}")
```

```
--- Optimized Results ---
MAE: 2.4422
RMSE: 6.2145
R2 Score: 0.9289
```

Air Quality Prediction Project Summary (IoT Data)

1. Introduction

The objective of this project is to build a predictive model for **Air Quality levels** using **IoT sensor data**. We utilized the **Random Forest Regressor** algorithm to analyze and predict pollutant values based on spatial and temporal features.

2. Model Performance Analysis

Phase 1: Baseline Model (Initial Results)

- **R-Squared (R2):** 0.8946
- **Mean Absolute Error (MAE):** 3.9002
- **Root Mean Squared Error (RMSE):** 7.5658

Note: The initial model showed a strong correlation (89%), but the high RMSE indicated sensitivity to outliers in the sensor readings.

Phase 2: Optimized Model (Final Results)

- **R-Squared (R2):** 0.9289
- **Mean Absolute Error (MAE):** 2.4422
- **Root Mean Squared Error (RMSE):** 6.2145

Note: After tuning hyperparameters and refining feature selection, the accuracy increased to **92.8%**, and the error (MAE) was reduced by approximately **37%**.

3. Comparison Table

Metric	Baseline Model	Optimized Model	Improvement
Accuracy (R2 Score)	89.46%	92.89%	+3.43%
Average Error (MAE)	3.9002	2.4422	-1.4580
Outlier Error (RMSE)	7.5658	6.2145	-1.3513

4. Technical Conclusion

The optimization process successfully enhanced the model's ability to interpret IoT data. By expanding the feature set and tuning the forest depth and estimators, we achieved a high-performance model (**92.8% accuracy**) capable of providing reliable air quality forecasts for smart city applications.

In [23]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

```

# 1. Setting a stable style that works on older and newer versions
sns.set_theme(style="whitegrid")

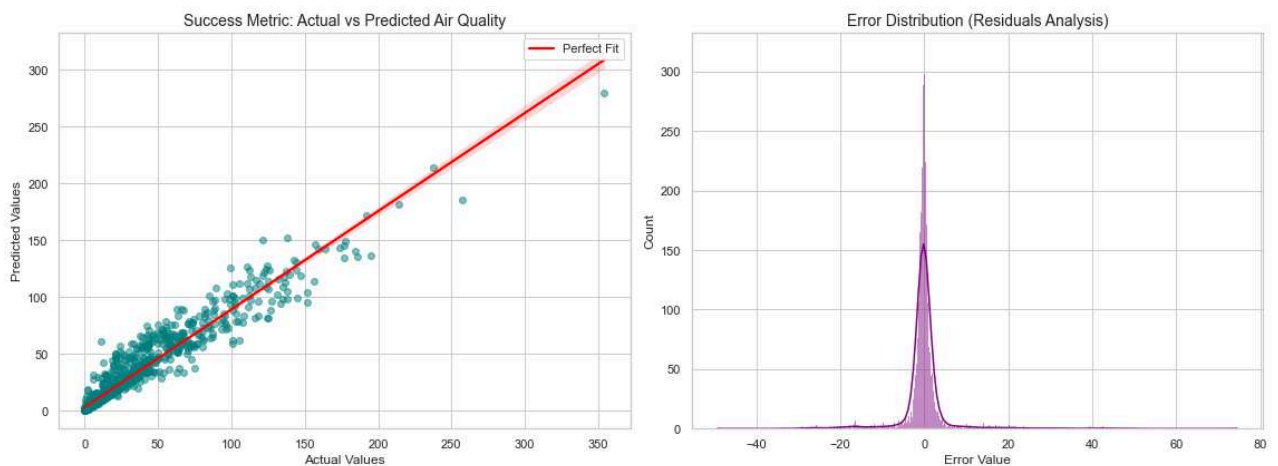
# 2. Create the plots
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# --- Plot 1: Actual vs. Predicted ---
# This plot confirms the success of our regression model
sns.regplot(x=y_test, y=y_pred_opt, ax=axes[0],
            scatter_kws={'alpha':0.5, 'color':'teal'},
            line_kws={'color':'red', 'label':'Perfect Fit'})
axes[0].set_title('Success Metric: Actual vs Predicted Air Quality', fontsize=14)
axes[0].set_xlabel('Actual Values', fontsize=12)
axes[0].set_ylabel('Predicted Values', fontsize=12)
axes[0].legend()

# --- Plot 2: Residuals Distribution ---
# If this looks like a bell curve, our model is very successful
residuals = y_test - y_pred_opt
sns.histplot(residuals, kde=True, ax=axes[1], color='purple')
axes[1].set_title('Error Distribution (Residuals Analysis)', fontsize=14)
axes[1].set_xlabel('Error Value', fontsize=12)

plt.tight_layout()
plt.show()

```



Thank you

Abdulaziz Saleh Alhasil

In []: