1. Linear Regression Model

Objective: Predict CO_2 emissions using time as the sole feature.

Data Loading and Preprocessing

python

Copy code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset
data =
pd.read_csv('C:/Users/subas/OneDrive/Desktop/MFC/cleaned_monthly_sectoral_data
set.csv')

# Convert 'Date' column to datetime
data['Date'] = pd.to_datetime(data['Date'])

# Convert 'Date' to numerical values (ordinal)
data['DateOrdinal'] = data['Date'].map(pd.Timestamp.toordinal)
```

- **Data Loading**: The dataset is read into a pandas DataFrame.
- **Date Conversion**: The 'Date' column is converted to datetime format to ensure proper handling.
- **Ordinal Encoding**: Dates are transformed into ordinal numbers (number of days since a fixed date) to serve as numerical features for the regression model.

Feature Selection and Normalization

python

Copy code

```
# Extract features and target variable
X = data[['DateOrdinal']]
y = data['Total Energy Electric Power Sector CO2 Emissions'].values.reshape(-1, 1)

# Normalize the target variable
scaler = MinMaxScaler()
y_normalized = scaler.fit_transform(y)
```

- **Feature Extraction**: 'DateOrdinal' is used as the independent variable.
- **Target Variable**: CO_2 emissions are the dependent variable.
- **Normalization**: The target variable is scaled to the [0, 1] range using MinMaxScaler to facilitate model training.

Data Splitting and Model Training

python

Copy code

```
# Split the data
X_train, X_test, y_train_normalized, y_test_normalized = train_test_split(
    X, y_normalized, test_size=0.15, random_state=42
)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train_normalized)
```

- **Data Splitting**: The dataset is divided into training and testing sets (85% training, 15% testing).
- **Model Training**: A linear regression model is trained on the training data.

Prediction and Evaluation

python

Copy code

```
# Predict
y_pred_normalized = model.predict(X_test)

# Metrics
mse = mean_squared_error(y_test_normalized, y_pred_normalized)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_normalized, y_pred_normalized)
r2 = r2_score(y_test_normalized, y_pred_normalized)

# Results
results_df = pd.DataFrame({
    'Metric': ['RMSE', 'MAE', 'MSE', 'R-squared'],
    'Value': [rmse, mae, mse, r2]
})
```

- **Prediction**: The model predicts CO_2 emissions on the test set.
- **Evaluation Metrics**:
    - **MSE**: Measures the average squared difference between actual and predicted values.
    - **RMSE**: Square root of MSE, providing error in the same units as the target variable.
    - **MAE**: Average absolute difference between actual and predicted values.
    - **R-squared**: Proportion of variance in the dependent variable predictable from the independent variable.

Visualization

python

Copy code

```
# Merge X_test with corresponding dates
X_test_with_dates = X_test.copy()
```

```python
X_test_with_dates['Date'] =
X_test_with_dates['DateOrdinal'].map(pd.Timestamp.fromordinal)


# Sort by Date for better plotting

X_test_with_dates['Actual'] = y_test_normalized

X_test_with_dates['Predicted'] = y_pred_normalized

X_test_with_dates.sort_values('Date', inplace=True)


# Plot

plt.figure(figsize=(10, 6))

plt.plot(X_test_with_dates['Date'], X_test_with_dates['Actual'], label='Actual
(Normalized)', color='blue')

plt.plot(X_test_with_dates['Date'], X_test_with_dates['Predicted'], label='Predicted
(Normalized)', color='red')

plt.xlabel('Date')

plt.ylabel('Normalized CO2 Emissions')

plt.title('Linear Regression Predictions vs Actual (Normalized)')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()


# Print results

print(results_df)
```

- **Data Preparation**: The test set is augmented with actual and predicted values for plotting.
- **Plotting**: A line plot compares actual and predicted normalized CO_2 emissions over time.

---

2. LightGBM Model

Objective: Utilize multiple features to predict CO_2 emissions using a gradient boosting framework.

Data Loading and Preprocessing

python

Copy code

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

from lightgbm import LGBMRegressor


# Load dataset

data =
pd.read_csv('C:/Users/subas/OneDrive/Desktop/MFC/cleaned_monthly_sectoral_data
set.csv')


# Convert 'Date' to datetime

data['Date'] = pd.to_datetime(data['Date'], format='%Y-%m-%d')


# Interpolate missing values linearly

data.interpolate(method='linear', inplace=True)
```

- **Data Loading**: The dataset is read into a DataFrame.
- **Date Conversion**: Ensures the 'Date' column is in datetime format.
- **Missing Value Handling**: Linear interpolation fills in missing values in the dataset.

Feature Engineering and Normalization

python

Copy code

```
# Define the target column
target_column = 'Total Energy Electric Power Sector CO2 Emissions'

# Separate features and target
features = data.drop(columns=['Date', target_column])
target = data[[target_column]]

# Clean feature column names for LightGBM compatibility
features.columns = features.columns.str.replace(r'[^\w\s]', '', regex=True).str.replace(' ',
'_')

# Normalize features
feature_scaler = StandardScaler()
features_scaled = feature_scaler.fit_transform(features)

# Normalize target (to 0–1 range)
target_scaler = MinMaxScaler()
target_scaled = target_scaler.fit_transform(target)
```

- **Feature Selection**: Excludes 'Date' and the target column from features.
- **Column Name Cleaning**: Removes special characters and spaces for compatibility with LightGBM.
- **Normalization**:
    - **Features**: StandardScaler standardizes features to have zero mean and unit variance.
    - **Target**: MinMaxScaler scales the target variable to the [0, 1] range.

Data Splitting and Model Training

python

Copy code

```
# Create DataFrames
X = pd.DataFrame(features_scaled, columns=features.columns)
y = pd.Series(target_scaled.flatten(), name=target_column)

# Train/test split (85% train, 15% test)
train_ratio = 0.85
train_size = int(len(X) * train_ratio)
X_train, X_test = X.iloc[:train_size], X.iloc[train_size:]
y_train, y_test = y.iloc[:train_size], y.iloc[train_size:]

# Train LightGBM Regressor
model = LGBMRegressor()
model.fit(X_train, y_train)
```

Splitting Data:

Rather than using train_test_split, here we split manually by index to preserve the time series order (important for temporal data like monthly CO_2).

85% of data is used for training, and the remaining 15% for testing.

Model Training:

LightGBM (Light Gradient Boosting Machine) is a fast, efficient implementation of gradient boosting.

It's trained using the training features and target.

Model Prediction and Evaluation

python

Copy code

```
# Predict and evaluate
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Display results
results_df = pd.DataFrame({
    'Metric': ['RMSE', 'MAE', 'MSE', 'R-squared'],
    'Value': [rmse, mae, mse, r2]
})
print(results_df)
```

Predictions: The model outputs normalized predicted CO_2 values.

Evaluation Metrics:

MSE (Mean Squared Error): Penalizes larger errors more than smaller ones.

RMSE (Root Mean Squared Error): Interpretable in the same unit as the target.

MAE (Mean Absolute Error): Average magnitude of prediction error.

R^2 (R-squared): Indicates how well the predictions match actual values. Closer to 1 means better fit.

Prediction Visualization

python

Copy code

```
# Prepare date data for plotting
dates = data['Date'].iloc[train_size:].reset_index(drop=True)

# Plot actual vs predicted
plt.figure(figsize=(14, 6))
plt.plot(dates, y_test, label='Actual (Normalized)', color='blue')
plt.plot(dates, y_pred, label='Predicted (Normalized)', color='red')
plt.xlabel('Date')
plt.ylabel('Normalized CO2 Emissions')
plt.title('LightGBM Model Predictions vs Actual')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Dates for X-axis: Corresponds to the test set period.

Line Plot:

Blue line: Actual normalized CO_2 emissions.

Red line: Model predictions.

This visualization helps us understand temporal prediction performance. 🔷 **XGBoost Regressor – Full Line-by-Line Breakdown**

python

Copy code

```
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100)
```

- xgb.XGBRegressor: Initializes an XGBoost regression model.
- objective='reg:squarederror': Specifies loss function (here: squared error for regression).
- n_estimators=100: Use 100 trees in the ensemble (more trees = higher capacity, more time).

python

Copy code

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

- Fits the model on training data. The model learns patterns by boosting weak learners (trees).

python

Copy code

```
y_pred = model.predict(X_test)
```

- Uses the trained model to make predictions on the test set.

python

Copy code

```
mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)
```

- mse: Mean Squared Error — average squared difference between actual and predicted.
- rmse: Root Mean Squared Error — square root of MSE (easier to interpret).
- mae: Mean Absolute Error — average of absolute differences.
- r2: R-squared — measures how well predictions approximate the real data (1 is perfect).

🔶 SVM (Support Vector Machine) – Full Line-by-Line Breakdown

python

Copy code

```
model = SVR(kernel='rbf')
```

- SVR: Support Vector Regressor — version of SVM for regression tasks.
- kernel='rbf': Radial Basis Function kernel — allows non-linear separation by projecting to a higher dimension.

python

Copy code

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

- Trains the SVM by trying to find a decision boundary that allows predictions within a margin of error, minimizing violations.

python

Copy code

```
y_pred = model.predict(X_test)
```

- Generates predictions for test inputs using the trained SVM model.

python

Copy code

```
mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)
```

- Same evaluation metrics as above to assess prediction quality.

🔷 LSTM (Long Short-Term Memory) – Full Deep Learning Breakdown

python

Copy code

X_lstm = X.values

y_lstm = y.values

- Converts pandas DataFrame to NumPy arrays. Neural networks require raw array formats.

python

Copy code

X_lstm = X_lstm.reshape((X_lstm.shape[0], 1, X_lstm.shape[1]))

- Reshapes data into 3D array: (samples, timesteps, features).
- For LSTM, timesteps=1 means we feed in one timestep at a time with multiple features.

python

Copy code

X_train_lstm, X_test_lstm = X_lstm[:train_size], X_lstm[train_size:]

y_train_lstm, y_test_lstm = y_lstm[:train_size], y_lstm[train_size:]

- Splits data chronologically into training and test sets, preserving time series integrity.

🧠 Building the LSTM Model

python

Copy code

model = Sequential()

- Sequential: A linear stack of layers — you define them one by one.

python

Copy code

model.add(LSTM(64, activation='relu', input_shape=(X_lstm.shape[1], X_lstm.shape[2])))

- Adds an LSTM layer:
  - 64: Number of memory cells/units.
  - activation='relu': Non-linearity to improve learning.

- input_shape=(1, num_features): 1 timestep, with multiple features.

python

Copy code

```
model.add(Dense(1))
```

- Final output layer: 1 neuron → outputs a single value (CO_2 emission prediction).

python

Copy code

```
model.compile(optimizer='adam', loss='mse')
```

- Compiles model:
  - optimizer='adam': Adaptive optimizer (adjusts learning rate).
  - loss='mse': Minimizes Mean Squared Error during training.

python

Copy code

```
model.fit(X_train_lstm, y_train_lstm, epochs=50, batch_size=16, verbose=1)
```

- Trains the model:
  - epochs=50: Model goes through data 50 times.
  - batch_size=16: Updates weights after every 16 samples.
  - verbose=1: Prints progress bar during training.

🔮 LSTM Predictions

python

Copy code

```
y_pred_lstm = model.predict(X_test_lstm)
```

- Predicts outputs from test inputs.

python

Copy code

```
mse = mean_squared_error(y_test_lstm, y_pred_lstm)

rmse = np.sqrt(mse)
```

```
mae = mean_absolute_error(y_test_lstm, y_pred_lstm)

r2 = r2_score(y_test_lstm, y_pred_lstm)
```

- Same evaluation metrics as earlier.