iot-carbon-emission-prediction

October 17, 2025

1 IoT-Carbon Footprint Dataset

This dataset contains IoT-based data designed to track and analyze the carbon footprint of individuals, based on various factors such as energy consumption, transportation activity, and environmental conditions. The dataset includes 10,000 entries, each representing an individual, with the following features:

1) Person_ID: A unique identifier for each individual (from 1 to 10,000).
2) Energy_Usage_kWh: The total daily energy consumption in kilowatt-hours, tracked via smart meters. Values range from 2 to 50 kWh.
3) Transportation_Distance_km: The total daily distance traveled by the individual in kilometers, tracked via GPS. Values range from 0 to 100 km.
4) Vehicle_Type: The mode of transportation used by the individual, with possible values: "Car", "Bus", "Walking", and "Electric Vehicle.
5) Smart_Appliance_Usage_hours: The daily usage (in hours) of smart appliances within the home, ranging from 1 to 12 hours.
6) Renewable_Energy_Usage_percent: The percentage of energy usage that comes from renewable sources, ranging from 0% to 100%.
7) Building_Type: The type of building where the individual resides, with possible values: "Residential" and "Commercial."
8) Temperature_C: The ambient temperature in Celsius, tracked by smart thermostats or weather stations, ranging from -10°C to 40°C.
9) Humidity_percent: The percentage of humidity, tracked by IoT humidity sensors, with values ranging from 20% to 90%.
10) Carbon_Emission_kgCO2: The estimated carbon emissions (in kilograms of CO) resulting from the individual's energy usage and transportation activities.

1.0.1 Data Preprocessing

[50]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
#Loading the dataset
data = pd.read_csv('IoT_Carbon_Footprint_Dataset.csv')
print(data.head())
```

```
   Person_ID  Energy_Usage_kWh  Transportation_Distance_km      Vehicle_Type  \
0          1         19.977926                   37.364082               Bus
1          2         47.634287                   33.291210               Bus
2          3         37.135709                   17.615391  Electric Vehicle
3          4         30.735607                   60.726667               Car
4          5          9.488895                   47.662416               Car

   Smart_Appliance_Usage_hours  Renewable_Energy_Usage_percent Building_Type  \
0                     2.016079                       60.939263    Commercial
1                     1.668291                        3.890347    Commercial
2                     7.646111                       61.226034   Residential
3                    11.627280                        8.966861    Commercial
4                     6.529934                       71.034424    Commercial

   Temperature_C  Humidity_percent  Carbon_Emission_kgCO2
0      32.361829         71.908864              18.012027
1      14.725852         81.677131              31.243122
2      -0.226719         52.422591              21.801932
3      26.832089         40.242511              30.353545
4      10.933907         42.319260              17.750117
```

[51]:
```python
#Unneeded column
data.drop('Person_ID',axis=1,inplace=True)
```

[52]:
```python
unique_columns = ['Vehicle_Type','Building_Type']
def show_unique(df,columns):
    for column in columns:
        print(df[column].unique())
show_unique(data,unique_columns)
```

```
['Bus' 'Electric Vehicle' 'Car' 'Walking']
['Commercial' 'Residential']
```

[53]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Energy_Usage_kWh                10000 non-null  float64
 1   Transportation_Distance_km      10000 non-null  float64
 2   Vehicle_Type                    10000 non-null  object
 3   Smart_Appliance_Usage_hours     10000 non-null  float64
```

```
 4   Renewable_Energy_Usage_percent  10000 non-null  float64
 5   Building_Type                   10000 non-null  object
 6   Temperature_C                   10000 non-null  float64
 7   Humidity_percent                10000 non-null  float64
 8   Carbon_Emission_kgCO2           10000 non-null  float64
dtypes: float64(7), object(2)
memory usage: 703.3+ KB
```

[54]:
```
data.describe()
```

[54]:
```
       Energy_Usage_kWh  Transportation_Distance_km  \
count      10000.000000                10000.000000
mean          25.719659                   50.452988
std           13.806246                   28.929455
min            2.000558                    0.015774
25%           13.823786                   25.394580
50%           25.641374                   50.589678
75%           37.520305                   75.647922
max           49.986448                   99.992483

       Smart_Appliance_Usage_hours  Renewable_Energy_Usage_percent  \
count                 10000.000000                    10000.000000
mean                      6.455659                       49.763431
std                       3.151162                       28.947451
min                       1.000061                        0.001674
25%                       3.721966                       24.681674
50%                       6.463859                       49.398761
75%                       9.133234                       74.974221
max                      11.997683                       99.997215

       Temperature_C  Humidity_percent  Carbon_Emission_kgCO2
count   10000.000000      10000.000000           10000.000000
mean       15.157229         55.125168              22.955986
std        14.417064         20.136054               9.738267
min        -9.999578         20.018909               1.186780
25%         2.847824         37.771932              15.734459
50%        15.304519         55.294661              22.623366
75%        27.672299         72.672843              29.533892
max        39.996985         89.978291              55.961625
```

[55]:
```
#Checking for null values if exists
data.isnull().sum()
```

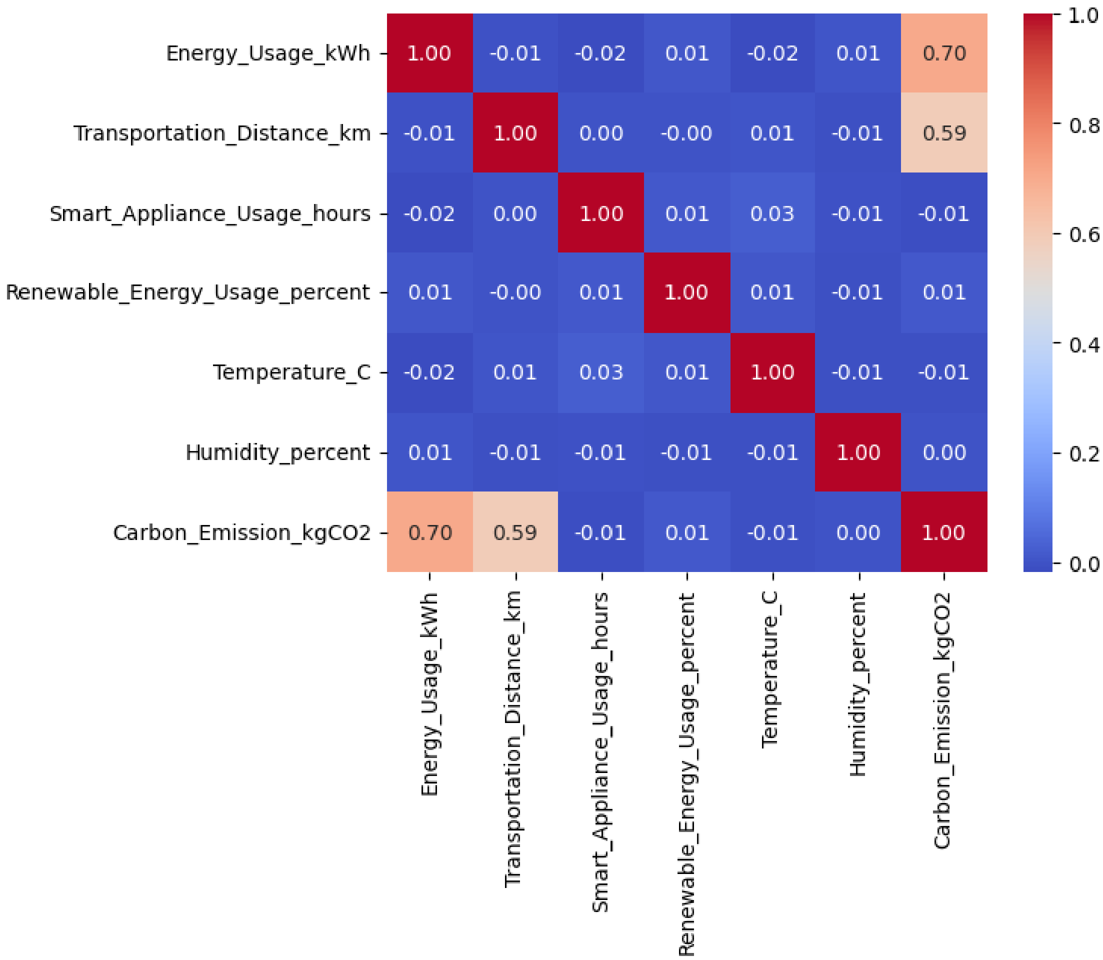
[55]:
```
Energy_Usage_kWh                  0
Transportation_Distance_km        0
Vehicle_Type                      0
Smart_Appliance_Usage_hours       0
```

```
Renewable_Energy_Usage_percent    0
Building_Type                     0
Temperature_C                     0
Humidity_percent                  0
Carbon_Emission_kgCO2             0
dtype: int64
```

1.0.2 Exploratory Data Analysis

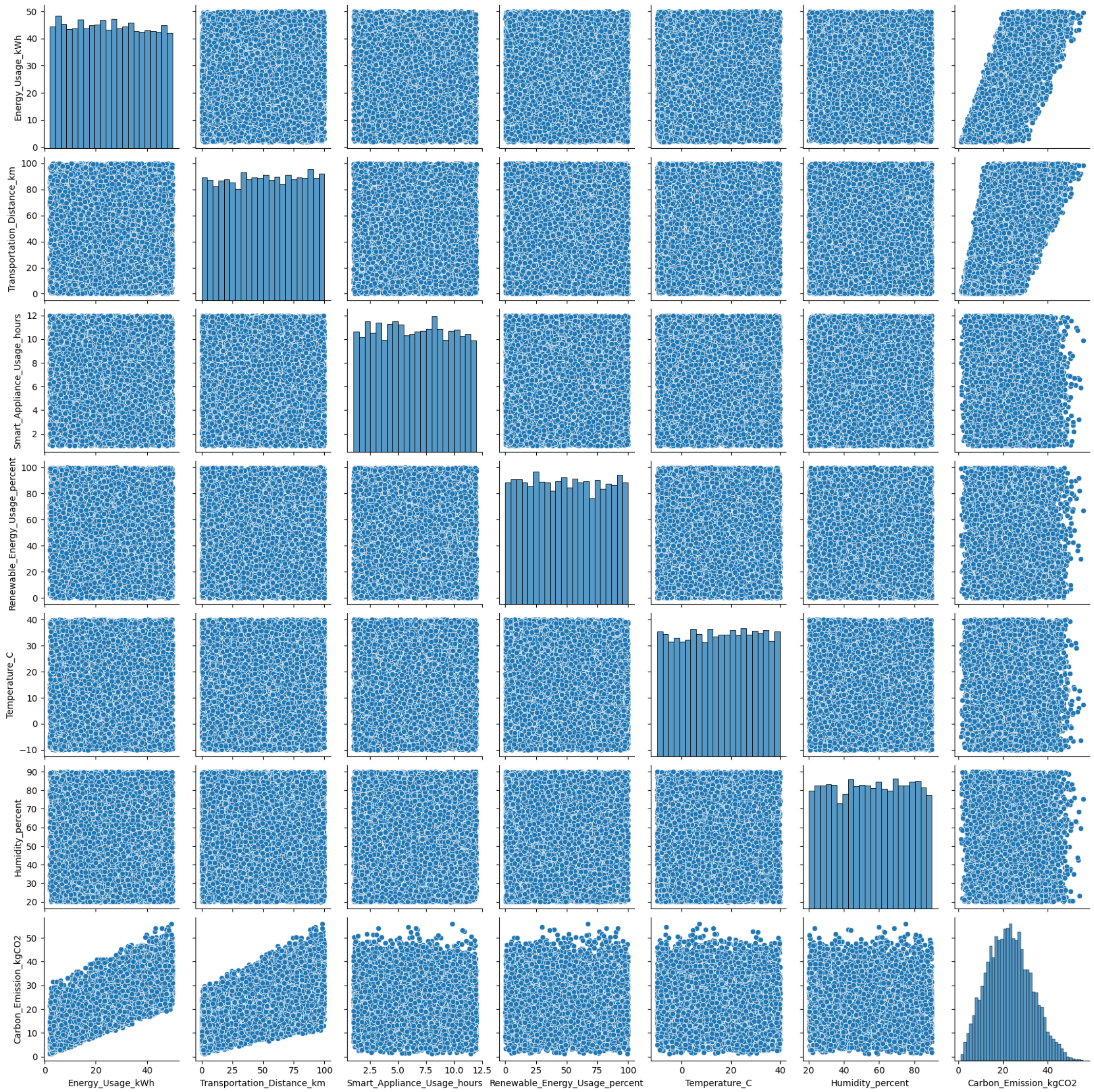
[56]:
```python
#Checking heatmap for correlations
correlation_matrix = data.corr(numeric_only=True)
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
```

[56]: <Axes: >



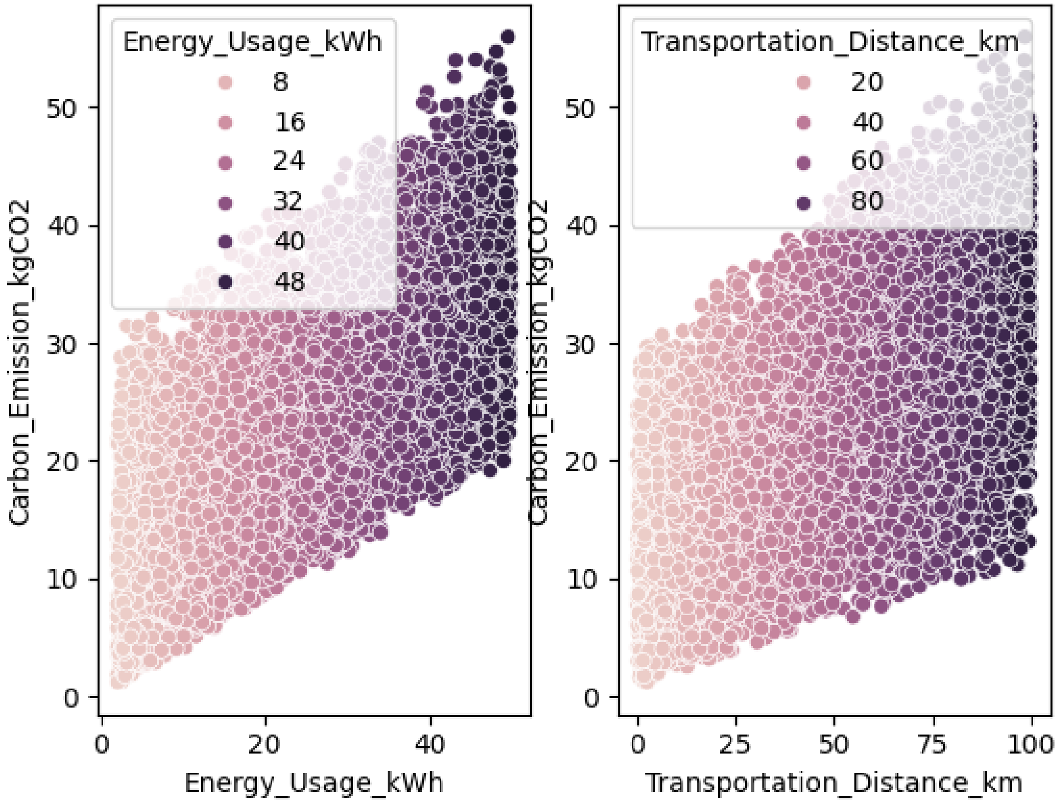
[57]:
```python
sns.pairplot(data=data)
```

[57]: <seaborn.axisgrid.PairGrid at 0x114dc1bd0>



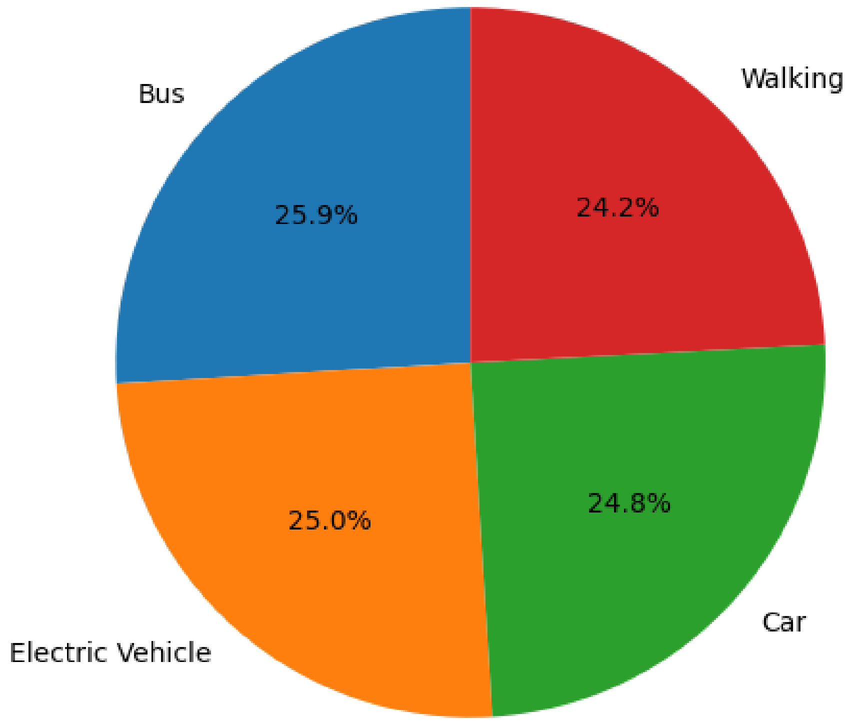
[58]:
```
plt.subplot(1,2,1)
sns.
  ↪scatterplot(data=data,x='Energy_Usage_kWh',y='Carbon_Emission_kgCO2',hue='Energy_Usage_kWh'
plt.subplot(1,2,2)
sns.
  ↪scatterplot(data=data,x='Transportation_Distance_km',y='Carbon_Emission_kgCO2',hue='Transpor
```

[58]: <Axes: xlabel='Transportation_Distance_km', ylabel='Carbon_Emission_kgCO2'>



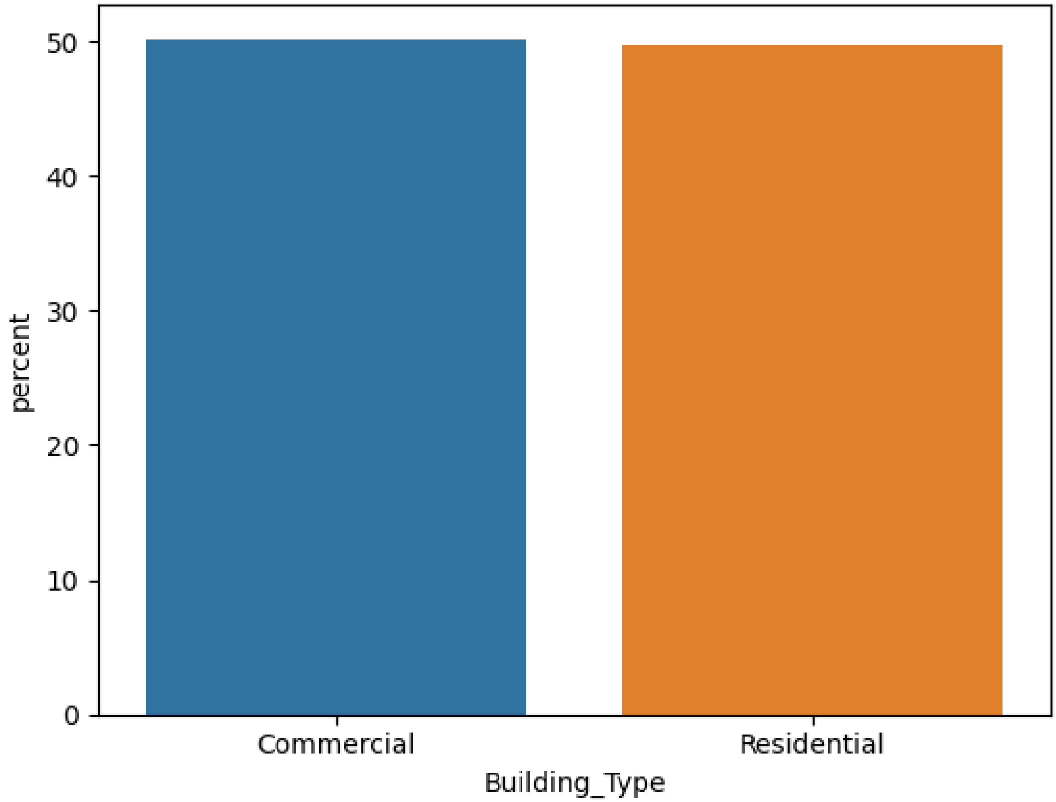
[59]:
```
plt.figure(figsize=(10,6))
plt.pie(data['Vehicle_Type'].value_counts(),labels=['Bus','Electric Vehicle',␣
↪'Car','Walking'], autopct='%1.1f%%', startangle=90)
```

[59]:
```
([<matplotlib.patches.Wedge at 0x1180339d0>,
  <matplotlib.patches.Wedge at 0x118033d90>,
  <matplotlib.patches.Wedge at 0x115588190>,
  <matplotlib.patches.Wedge at 0x115588550>],
 [Text(-0.8006817744982567, 0.7542603635259664, 'Bus'),
  Text(-0.7295142647923714, -0.8232915264136064, 'Electric Vehicle'),
  Text(0.8198447959613582, -0.7333856492562962, 'Car'),
  Text(0.7580252428275027, 0.7971183922331148, 'Walking')],
 [Text(-0.43673551336268546, 0.41141474374143616, '25.9%'),
  Text(-0.3979168717049298, -0.44906810531651253, '25.0%'),
  Text(0.44718807052437715, -0.40002853595797977, '24.8%'),
  Text(0.41346831426954694, 0.43479185030897166, '24.2%')])
```



[60]:
```
sns.countplot(data=data,x='Building_Type',hue='Building_Type',stat='percent')
```

[60]: <Axes: xlabel='Building_Type', ylabel='percent'>



1.0.3 Data Preprocessing Part 2

[61]:
```
#One hot encoding for Vehicles
vehicles = pd.get_dummies(data['Vehicle_Type'],drop_first=True).astype(int)
data.drop('Vehicle_Type',axis=1,inplace=True)
data = pd.concat([data,vehicles],axis=1)
```

[62]:
```
#Encoding Building Type
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data['Building_Type'] = encoder.fit_transform(data['Building_Type'])
```

[63]:
```
data.head()
```

[63]:
```
   Energy_Usage_kWh  Transportation_Distance_km  Smart_Appliance_Usage_hours  \
0         19.977926                   37.364082                     2.016079
1         47.634287                   33.291210                     1.668291
2         37.135709                   17.615391                     7.646111
3         30.735607                   60.726667                    11.627280
4          9.488895                   47.662416                     6.529934
```

```
   Renewable_Energy_Usage_percent  Building_Type  Temperature_C  \
0                       60.939263              0      32.361829
1                        3.890347              0      14.725852
2                       61.226034              1      -0.226719
3                        8.966861              0      26.832089
4                       71.034424              0      10.933907

   Humidity_percent  Carbon_Emission_kgCO2  Car  Electric Vehicle  Walking
0         71.908864              18.012027    0                 0        0
1         81.677131              31.243122    0                 0        0
2         52.422591              21.801932    0                 1        0
3         40.242511              30.353545    1                 0        0
4         42.319260              17.750117    1                 0        0
```

[64]:
```
#Preparing data for model
X = data.drop('Carbon_Emission_kgCO2',axis=1)
y = data['Carbon_Emission_kgCO2']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
y = scaler.fit_transform(y.values.reshape(-1,1))
```

1.0.4 Train Test Split

[65]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7)
y_train = y_train.ravel()
y_test = y_test.ravel()
```

1.0.5 Linear Regression

[66]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression().fit(X_train,y_train)
lr.score(X_train,y_train)
```

[66]: 0.8483911589662304

[67]:
```
lr_preds = lr.predict(X_test)
```

1.0.6 Polynomial Regression

[68]:
```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=2)
#Transforming the features to higher degree
X_train_poly = poly_reg.fit_transform(X_train)
```

```
X_test_poly = poly_reg.transform(X_test)
```

[69]:
```
plr = LinearRegression()
plr.fit(X_train_poly,y_train)
#Model accuracy
plr.score(X_train_poly,y_train)
```

[69]: 0.8492921458919694

[70]:
```
plr_preds = plr.predict(X_test_poly)
```

1.0.7 Random Forest Regressor

[71]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor().fit(X_train,y_train)
rf.score(X_train,y_train)
```

[71]: 0.9766044190347396

[72]:
```
rf_preds = rf.predict(X_test)
```

1.0.8 Support Vector Regressor (SVR)

[73]:
```
from sklearn.svm import SVR
svr = SVR().fit(X_train,y_train)
svr.score(X_train,y_train)
```

[73]: 0.8486348524461467

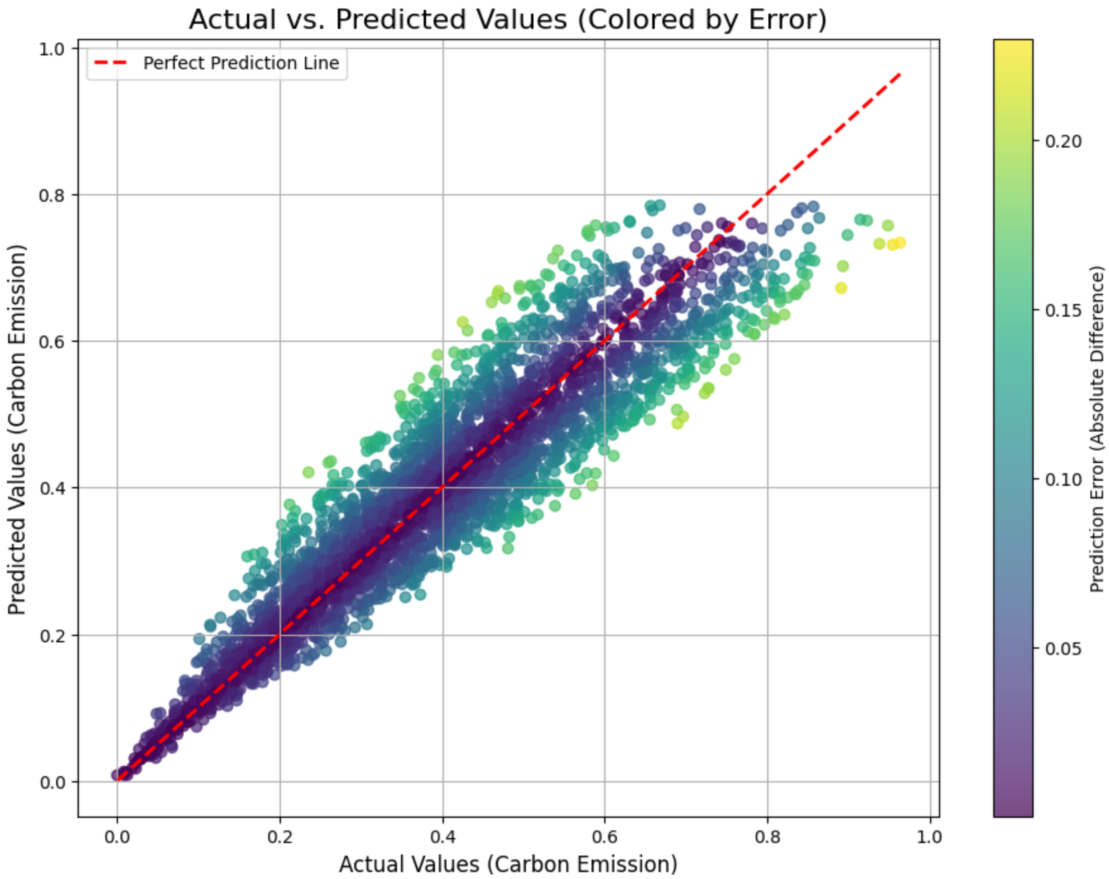
[74]:
```
svr_preds = svr.predict(X_test)
```

1.0.9 Model Evaluation

[75]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

[76]:
```
#Acutal vs Predicted Values for Linear Regression
plt.figure(figsize=(11, 8))

# Calculate the absolute error for each point
errors = np.abs(y_test - lr_preds)

# Create the scatter plot, coloring the points by their error
points = plt.scatter(y_test, lr_preds, c=errors, cmap='viridis', alpha=0.7)

# Add a color bar to the side to act as a legend for the error values
plt.colorbar(points, label='Prediction Error (Absolute Difference)')
```

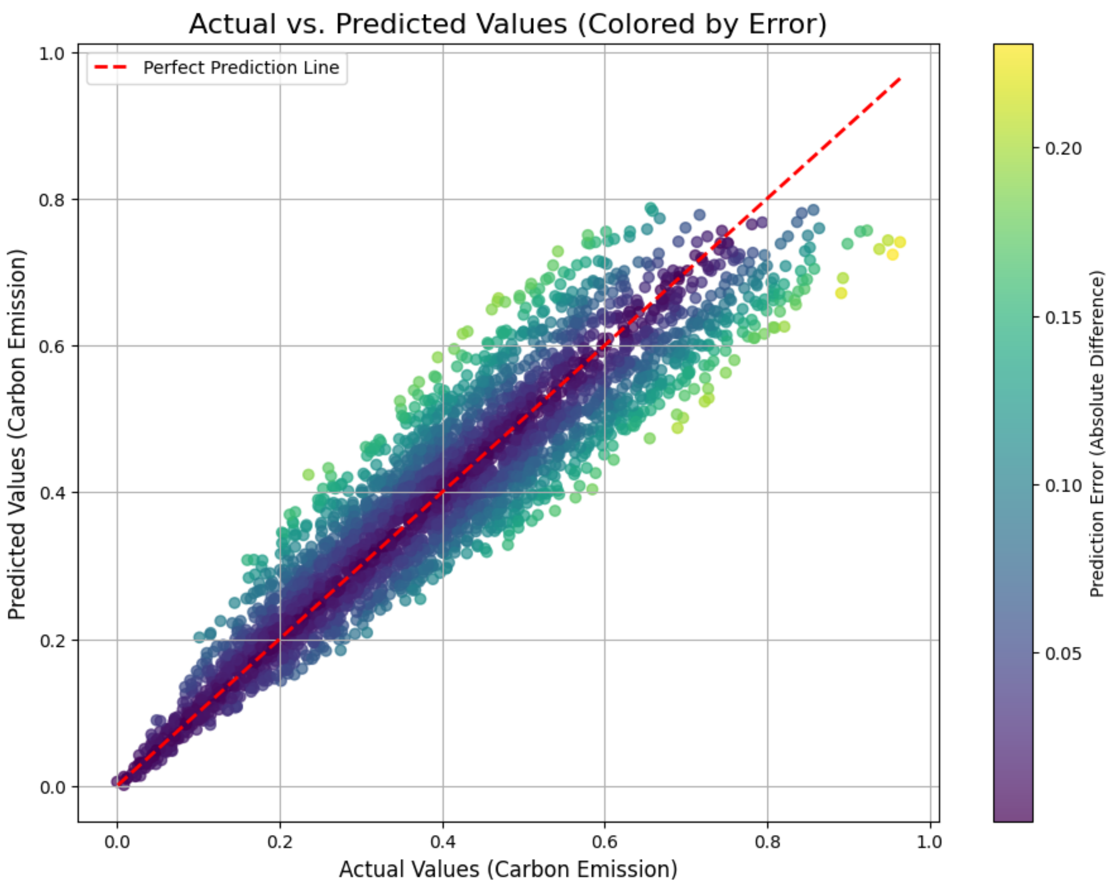
```python
# Plot the perfect prediction line (y=x) for reference
p1 = max(max(lr_preds), max(y_test))
p2 = min(min(lr_preds), min(y_test))
plt.plot([p1, p2], [p1, p2], 'r--', linewidth=2, label='Perfect Prediction␣
↪Line')

plt.title('Actual vs. Predicted Values (Colored by Error)', fontsize=16)
plt.xlabel('Actual Values (Carbon Emission)', fontsize=12)
plt.ylabel('Predicted Values (Carbon Emission)', fontsize=12)
plt.legend()
plt.grid(True)
```



[77]:
```python
print("Linear Regression:")
print("MSE:", mean_squared_error(y_test, lr_preds))
print("MAE:", mean_absolute_error(y_test, lr_preds))
print("R²:", r2_score(y_test, lr_preds))
print('Accuracy:', lr.score(X_test,y_test))
```

```
Linear Regression:
MSE: 0.004775877576193402
MAE: 0.05420508202669648
R²: 0.8498806574946713
Accuracy: 0.8498806574946713
```

[78]:
```python
#Actual vs Predicted Values for Polynomial Regression
plt.figure(figsize=(11, 8))

# Calculate the absolute error for each point
errors = np.abs(y_test - plr_preds)

# Create the scatter plot, coloring the points by their error
points = plt.scatter(y_test, plr_preds, c=errors, cmap='viridis', alpha=0.7)

# Add a color bar to the side to act as a legend for the error values
plt.colorbar(points, label='Prediction Error (Absolute Difference)')

# Plot the perfect prediction line (y=x) for reference
p1 = max(max(plr_preds), max(y_test))
p2 = min(min(plr_preds), min(y_test))
plt.plot([p1, p2], [p1, p2], 'r--', linewidth=2, label='Perfect Prediction␣
 ↪Line')

plt.title('Actual vs. Predicted Values (Colored by Error)', fontsize=16)
plt.xlabel('Actual Values (Carbon Emission)', fontsize=12)
plt.ylabel('Predicted Values (Carbon Emission)', fontsize=12)
plt.legend()
plt.grid(True)
plt.show()
```

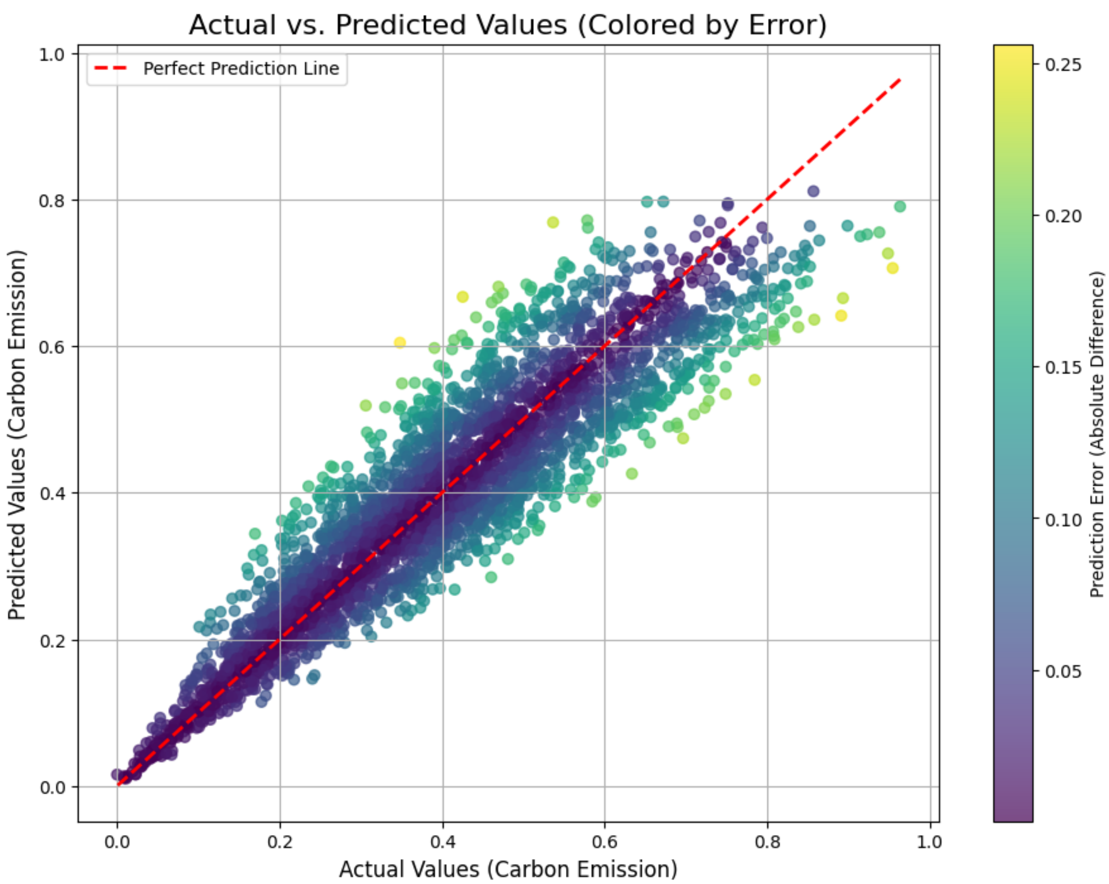


[79]:
```
print("Polynomial Regression:")
print("MSE:", mean_squared_error(y_test, plr_preds))
print("MAE:", mean_absolute_error(y_test, plr_preds))
print("R²:", r2_score(y_test, plr_preds))
print('Accuracy:', plr.score(X_test_poly,y_test))
```

```
Polynomial Regression:
MSE: 0.004819486230336618
MAE: 0.05449088813116564
R²: 0.8485099141322044
Accuracy: 0.8485099141322044
```

[80]:
```
#Acutal vs Predicted Values for Random Forest Regressor
plt.figure(figsize=(11, 8))

# Calculate the absolute error for each point
errors = np.abs(y_test - rf_preds)

# Create the scatter plot, coloring the points by their error
```

```
points = plt.scatter(y_test, rf_preds, c=errors, cmap='viridis', alpha=0.7)

# Add a color bar to the side to act as a legend for the error values
plt.colorbar(points, label='Prediction Error (Absolute Difference)')

# Plot the perfect prediction line (y=x) for reference
p1 = max(max(rf_preds), max(y_test))
p2 = min(min(rf_preds), min(y_test))
plt.plot([p1, p2], [p1, p2], 'r--', linewidth=2, label='Perfect Prediction␣
↪Line')

plt.title('Actual vs. Predicted Values (Colored by Error)', fontsize=16)
plt.xlabel('Actual Values (Carbon Emission)', fontsize=12)
plt.ylabel('Predicted Values (Carbon Emission)', fontsize=12)
plt.legend()
plt.grid(True)
```



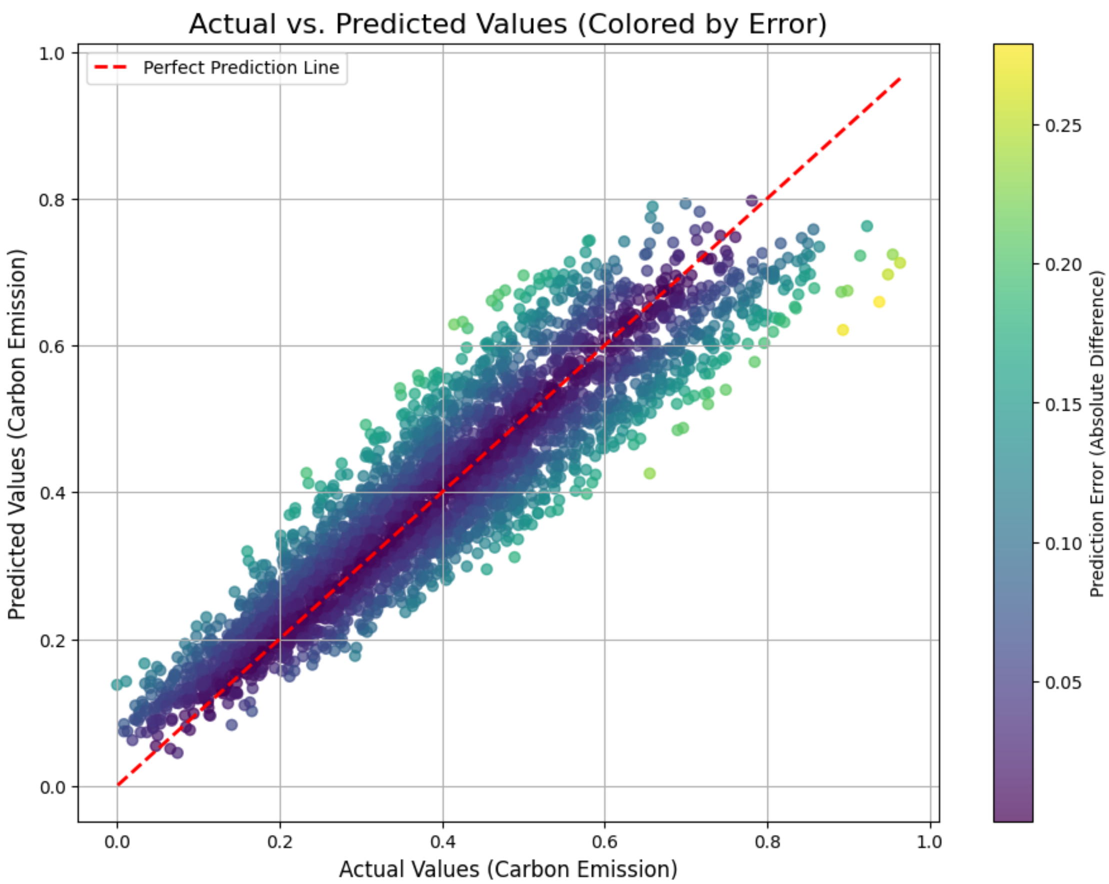
[81]:
```
print("Random Forest Regressior:")
print("MSE:", mean_squared_error(y_test, rf_preds))
```

```
print("MAE:", mean_absolute_error(y_test, rf_preds))
print("R²:", r2_score(y_test, rf_preds))
print('Accuracy:', rf.score(X_test,y_test))
```

```
Random Forest Regressior:
MSE: 0.005303066790795315
MAE: 0.05661495271565243
R²: 0.8333096091356347
Accuracy: 0.8333096091356347
```

[82]:

```
#Actual vs Predicted Values for SVR
plt.figure(figsize=(11, 8))

# Calculate the absolute error for each point
errors = np.abs(y_test - svr_preds)

# Create the scatter plot, coloring the points by their error
points = plt.scatter(y_test, svr_preds, c=errors, cmap='viridis', alpha=0.7)

# Add a color bar to the side to act as a legend for the error values
plt.colorbar(points, label='Prediction Error (Absolute Difference)')

# Plot the perfect prediction line (y=x) for reference
p1 = max(max(svr_preds), max(y_test))
p2 = min(min(svr_preds), min(y_test))
plt.plot([p1, p2], [p1, p2], 'r--', linewidth=2, label='Perfect Prediction␣
  ↪Line')

plt.title('Actual vs. Predicted Values (Colored by Error)', fontsize=16)
plt.xlabel('Actual Values (Carbon Emission)', fontsize=12)
plt.ylabel('Predicted Values (Carbon Emission)', fontsize=12)
plt.legend()
plt.grid(True)
```



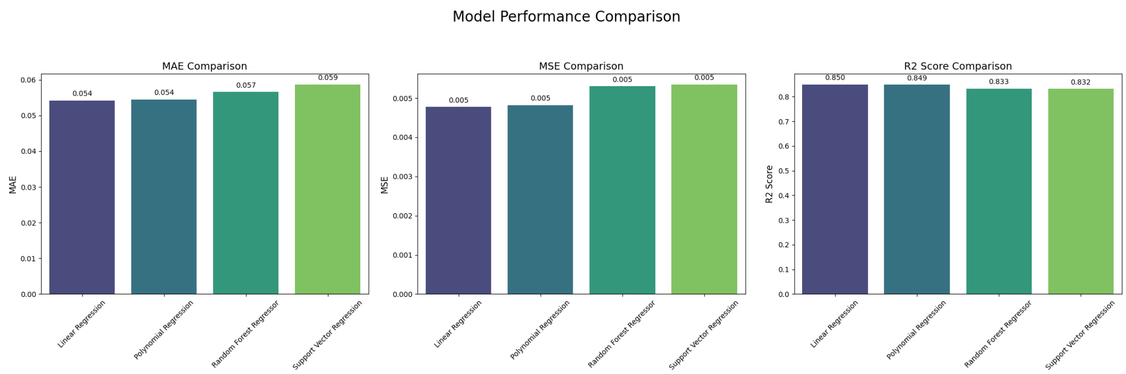
[83]:
```
print("SVR:")
print("MSE:", mean_squared_error(y_test, svr_preds))
print("MAE:", mean_absolute_error(y_test, svr_preds))
print("R²:", r2_score(y_test, svr_preds))
print('Accuracy:', svr.score(X_test,y_test))
```

```
SVR:
MSE: 0.005348874998450225
MAE: 0.058719297444641445
R²: 0.8318697275840679
Accuracy: 0.8318697275840679
```

1.0.10 Comparing Models

[84]:
```
#Getting data
results = pd.DataFrame({
    'Model': ['Linear Regression', 'Polynomial Regression', 'Random Forest␣
↪Regressor', 'Support Vector Regression'],
```

```
    'MAE': [mean_absolute_error(y_test, lr_preds), mean_absolute_error(y_test,␣
↪plr_preds), mean_absolute_error(y_test, rf_preds),␣
↪mean_absolute_error(y_test, svr_preds)],
    'MSE': [mean_squared_error(y_test, lr_preds), mean_squared_error(y_test,␣
↪plr_preds), mean_squared_error(y_test, rf_preds), mean_squared_error(y_test,␣
↪svr_preds)],
    'R2 Score': [r2_score(y_test, lr_preds), r2_score(y_test, plr_preds),␣
↪r2_score(y_test, rf_preds), r2_score(y_test, svr_preds)]
})

#Creating Graphics
fig, axes = plt.subplots(1, 3, figsize=(22, 7)) # Daha uygun bir en-boy oranı
fig.suptitle('Model Performance Comparison', fontsize=20, y=1.02)


metrics = ['MAE', 'MSE', 'R2 Score']
sorting_orders = [True, True, False]

for i, (metric, asc_order) in enumerate(zip(metrics, sorting_orders)):
    sorted_results = results.sort_values(by=metric, ascending=asc_order)

    #Barplot
    sns.barplot(x='Model', y=metric, data=sorted_results, ax=axes[i],
                palette='viridis', hue='Model', legend=False)

    for p in axes[i].patches:
        axes[i].annotate(format(p.get_height(), '.3f'),
                       (p.get_x() + p.get_width() / 2., p.get_height()),
                       ha = 'center', va = 'center',
                       xytext = (0, 9),
                       textcoords = 'offset points')

    axes[i].set_title(f'{metric} Comparison', fontsize=14)
    axes[i].set_xlabel('')
    axes[i].set_ylabel(metric, fontsize=12)
    axes[i].tick_params(axis='x', rotation=45)

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



1.1 Conclusion

The aim of this project was to predict the carbon emission (in kilograms of CO2) according to the given parameters. From the exploratory data analysis, it was obvious that the carbon emission is highly correlated with the consumption of energy and transportation usage.

Coming to the machine learning models, the linear regression was the best among the other models, with accuracy of %85. The polynomial regression and random forest regressor had accuracy of %84.8 and %83.3 respectively. The support vector regression (SVR) had the lowest accuracy of %83.2.