Linear Regression

In [27]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [6]:
```python
df1 = pd.read_csv('D:\manish\Python\GROW AI\python\Electric_Vehicle_Population_Data.csv')
df1
```

Out[6]:

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	Electric Range	Legislative District	DOL Vehicle ID	Vehicle Location	Electric Utility	2020 Census Tract
0	5YJYGDEE8L	Thurston	Tumwater	WA	98501.0	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	291.0	35.0	124633715	POINT (-122.89165 47.03954)	PUGET SOUND ENERGY INC	5.306701e+10
1	5YJXCAE2XJ	Snohomish	Bothell	WA	98021.0	2018	TESLA	MODEL X	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	238.0	1.0	474826075	POINT (-122.18384 47.8031)	PUGET SOUND ENERGY INC	5.306105e+10
2	5YJ3E1EBXK	King	Kent	WA	98031.0	2019	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	220.0	47.0	280307233	POINT (-122.17743 47.41185)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	5.303303e+10
3	7SAYGDEE4T	King	Issaquah	WA	98027.0	2026	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	0.0	41.0	280786565	POINT (-122.03439 47.5301)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	5.303302e+10
4	WAUUPBFF9G	King	Seattle	WA	98103.0	2016	AUDI	A3	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	16.0	43.0	198988891	POINT (-122.35436 47.67596)	CITY OF SEATTLE - (WA)|CITY OF TACOMA - (WA)	5.303300e+10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
270257	1C4RJXN60R	Pierce	Joint Base Lewis Mcchord	WA	98433.0	2024	JEEP	WRANGLER	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	21.0	28.0	266021122	POINT (-122.60761 47.08798)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	5.305307e+10
270258	1C4JJXR66N	Mason	Hoodsport	WA	98548.0	2022	JEEP	WRANGLER	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	22.0	35.0	282482938	POINT (-123.14135 47.40639)	BONNEVILLE POWER ADMINISTRATION||CITY OF TACOM...	5.304596e+10
270259	7SAYGDEEXP	Pierce	Tacoma	WA	98406.0	2023	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	0.0	27.0	228485085	POINT (-122.52082 47.26887)	BONNEVILLE POWER ADMINISTRATION||CITY OF TACOM...	5.305306e+10
270260	5YJYGDEE2M	Snohomish	Bothell	WA	98021.0	2021	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	0.0	1.0	282699217	POINT (-122.18384 47.8031)	PUGET SOUND ENERGY INC	5.306105e+10
270261	JN1BF0BA5P	Chelan	Wenatchee	WA	98801.0	2023	NISSAN	ARIYA HATCHBACK	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	0.0	12.0	261475224	POINT (-120.30521 47.41493)	PUD NO 1 OF CHELAN COUNTY	5.300796e+10

270262 rows × 16 columns

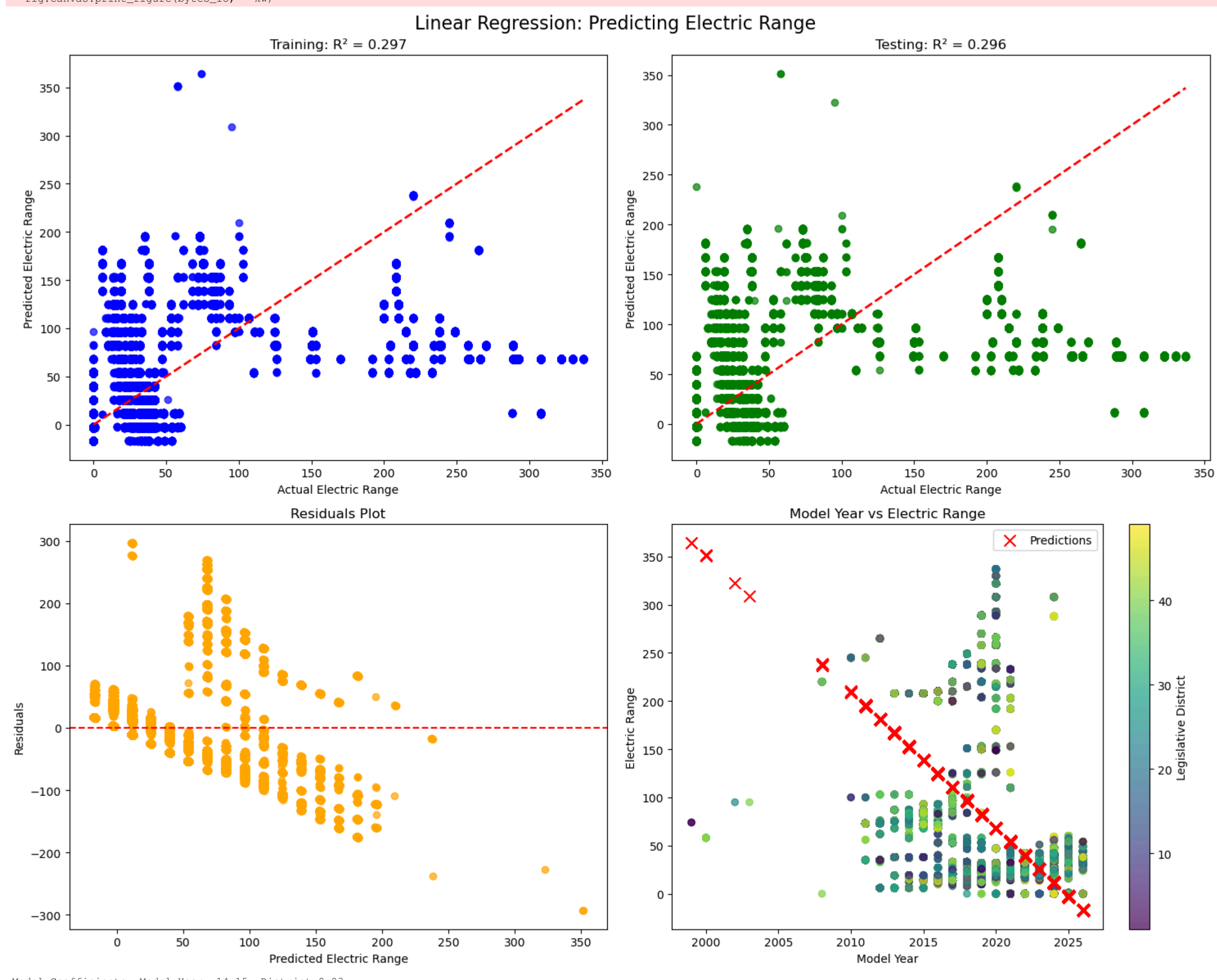
How can we use Linear Regression to predict the Electric Range of a vehicle?

In [8]:
```python
d1 = df[['Model Year', 'Electric Range', 'Legislative District']].dropna()
X = d1[['Model Year', 'Legislative District']]
y = d1['Electric Range']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("R²:", r2_score(y_test, y_pred))
print("Predictions:", y_pred)
```

```
Coefficients: [-14.14570503   0.02866648]
Intercept: 28641.659603274988
R²: 0.295898700156663
Predictions: [ 12.04262164  40.36269818  25.24233268 ... -16.22012193  -2.21774932
  39.30203825]
```

In [10]:
```python
d1 = df[['Model Year', 'Electric Range', 'Legislative District']].dropna()
X = d1[['Model Year', 'Legislative District']]
y = d1['Electric Range']

# Train model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Create visualizations
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
fig.suptitle('Linear Regression: Predicting Electric Range', fontsize=16)

# 1. Actual vs Predicted (Training)
axes[0,0].scatter(y_train, y_pred_train, alpha=0.7, color='blue')
axes[0,0].plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'r--', lw=2)
axes[0,0].set_xlabel('Actual Electric Range')
axes[0,0].set_ylabel('Predicted Electric Range')
axes[0,0].set_title(f'Training: R² = {r2_score(y_train, y_pred_train):.3f}')

# 2. Actual vs Predicted (Testing)
axes[0,1].scatter(y_test, y_pred_test, alpha=0.7, color='green')
axes[0,1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[0,1].set_xlabel('Actual Electric Range')
axes[0,1].set_ylabel('Predicted Electric Range')
axes[0,1].set_title(f'Testing: R² = {r2_score(y_test, y_pred_test):.3f}')

# 3. Residuals Plot
residuals = y_test - y_pred_test
axes[1,0].scatter(y_pred_test, residuals, alpha=0.7, color='orange')
axes[1,0].axhline(y=0, color='r', linestyle='--')
axes[1,0].set_xlabel('Predicted Electric Range')
axes[1,0].set_ylabel('Residuals')
axes[1,0].set_title('Residuals Plot')

# 4. Model Year vs Electric Range with regression line
scatter = axes[1,1].scatter(d1['Model Year'], d1['Electric Range'],
                           c=d1['Legislative District'], cmap='viridis', alpha=0.7)
model_year_line = axes[1,1].scatter(X['Model Year'], model.predict(X),
                                   color='red', s=100, marker='x', label='Predictions')
axes[1,1].set_xlabel('Model Year')
axes[1,1].set_ylabel('Electric Range')
axes[1,1].set_title('Model Year vs Electric Range')
plt.colorbar(scatter, ax=axes[1,1], label='Legislative District')
axes[1,1].legend()

plt.tight_layout()
plt.show()

# Print model statistics
print(f"Model Coefficients: Model Year={model.coef_[0]:.2f}, District={model.coef_[1]:.2f}")
print(f"Intercept: {model.intercept_:.2f}")
print(f"Test R²: {r2_score(y_test, y_pred_test):.3f}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_test)):.2f}")
```

```
C:\Users\LENOVO\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```




```
Model Coefficients: Model Year=-14.15, District=0.03
Intercept: 28641.66
Test R²: 0.296
RMSE: 66.52
```

What independent variables (features) can be used to predict Electric Range? (e.g., Model Year, Base MSRP, Make)

In [11]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder

# Potential numeric features for Electric Range prediction
numeric_features = ['Model Year', 'Legislative District', 'Postal Code']
categorical_features = ['Make', 'Model', 'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAFV) Eligibility']

# Create feature matrix (encode categoricals, handle missing values)
df_numeric = df[numeric_features + ['Electric Range']].dropna()
X_numeric = df_numeric[numeric_features]

# Encode categorical features (one-hot for small dataset)
df_cat = pd.get_dummies(df[categorical_features + ['Electric Range']].dropna(), drop_first=True)
X_cat = df_cat.drop('Electric Range', axis=1)

print("Available Numeric Features:", numeric_features)
print("Available Categorical Features:", categorical_features)
print("\nDataset shape after cleaning:", df_numeric.shape)
```

```
Available Numeric Features: ['Model Year', 'Legislative District', 'Postal Code']
Available Categorical Features: ['Make', 'Model', 'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAFV) Eligibility']

Dataset shape after cleaning: (269608, 4)
```

How do we handle categorical variables like Make and Model in regression analysis?

In [14]:
```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

df_clean = df1[['Make', 'Model', 'Electric Range', 'Model Year']].dropna()

# METHOD 1: ONE-HOT ENCODING (Recommended for nominal categoricals)
X_cat = df_clean[['Make', 'Model']]
y = df_clean['Electric Range']

ohe = OneHotEncoder(drop='first', sparse_output=False)  # drop='first' avoids multicollinearity
X_ohe = ohe.fit_transform(X_cat)
ohe_feature_names = ohe.get_feature_names_out(['Make', 'Model'])
X_ohe_df = pd.DataFrame(X_ohe, columns=ohe_feature_names)

print("One-Hot Encoded features shape:", X_ohe_df.shape)
print("Sample columns:", X_ohe_df.columns[:5].tolist())
```

```
One-Hot Encoded features shape: (270257, 228)
Sample columns: ['Make_ALFA ROMEO', 'Make_AUDI', 'Make_AZURE DYNAMICS', 'Make_BENTLEY', 'Make_BMW']
```

In [17]:
```python
# Combine with numeric features
X_numeric = df_clean[['Model Year']]
X_final = pd.concat([X_numeric.reset_index(drop=True), X_ohe_df.reset_index(drop=True)], axis=1)

# Train-test split and model
X_train, X_test, y_train, y_test = train_test_split(X_final, y.reset_index(drop=True),
                                                    test_size=0.3, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f"R² Score: {r2_score(y_test, y_pred):.3f}")
print("Top Make coefficients:")
make_coefs = {col: coef for col, coef in zip(X_final.columns, model.coef_) if 'Make_' in col}
print(sorted(make_coefs.items(), key=lambda x: x[1], reverse=True)[:3])

# METHOD 2: LABEL ENCODING (Alternative, simpler but assumes order)
le_make = LabelEncoder()
le_model = LabelEncoder()
df_clean['Make_encoded'] = le_make.fit_transform(df_clean['Make'])
df_clean['Model_encoded'] = le_model.fit_transform(df_clean['Model'])

X_label = df_clean[['Model Year', 'Make_encoded', 'Model_encoded']]
X_train_label, X_test_label, y_train, y_test = train_test_split(X_label, y, test_size=0.3, random_state=42)
model_label = LinearRegression().fit(X_train_label, y_train)
print(f"Label Encoding R²: {r2_score(y_test, model_label.predict(X_test_label)):.3f}")
```

```
R² Score: 0.513
Top Make coefficients:
[('Make_JAGUAR', np.float64(67.72202630810501)), ('Make_LAND ROVER', np.float64(38.050676780855014)), ('Make_BENTLEY', np.float64(37.17683097158081))]
Label Encoding R²: 0.304
```

What is the R² score of the model, and what does it indicate about prediction accuracy?

In [18]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

d1 = df1[['Model Year', 'Electric Range', 'Legislative District']].dropna()
X = d1[['Model Year', 'Legislative District']]
y = d1['Electric Range']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)

print(f"R² Score: {r2:.4f}")
```

```
R² Score: 0.2959
```

What steps are needed to improve the accuracy of the Linear Regression model?

In [20]:
```python
# Add categorical features via one-hot encoding
df_clean = df1[['Model Year', 'Make', 'Electric Vehicle Type', 'Electric Range']].dropna()

# One-hot encode Make and Vehicle Type
X = pd.get_dummies(df_clean[['Model Year', 'Make', 'Electric Vehicle Type']], drop_first=True)
y = df_clean['Electric Range']
```

In [21]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score

# Enhanced feature set
features = ['Model Year', 'Make', 'Electric Vehicle Type',
           'Clean Alternative Fuel Vehicle (CAFV) Eligibility', 'Legislative District']
df_enhanced = pd.get_dummies(df[features + ['Electric Range']].dropna(), drop_first=True)
X = df_enhanced.drop('Electric Range', axis=1)
y = df_enhanced['Electric Range']

# Scale features and split
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Compare models
models = {
    'Linear': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    r2 = r2_score(y_test, model.predict(X_test))
    cv_score = cross_val_score(model, X_train, y_train, cv=3).mean()
    print(f"{name} - Test R²: {r2:.3f}, CV R²: {cv_score:.3f}")
```

```
Linear - Test R²: 0.914, CV R²: 0.915
Ridge - Test R²: 0.914, CV R²: 0.915
Lasso - Test R²: 0.914, CV R²: 0.914
```

Can we use this model to predict the range of new EV models based on their specifications?

In [26]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score
import warnings; warnings.filterwarnings('ignore')


print(f"Original shape: {df.shape}")
print("NaN in Electric Range:", df['Electric Range'].isna().sum())

# **CRITICAL FIX 1: Remove rows where target (y) is NaN**
df_clean = df.dropna(subset=['Electric Range'])  # Drop rows with NaN target
print(f"Clean shape (y fixed): {df_clean.shape}")

# Define features
numeric_features = ['Model Year', 'Legislative District']
categorical_features = ['Make', 'Electric Vehicle Type',
                        'Clean Alternative Fuel Vehicle (CAFV) Eligibility']

# **CRITICAL FIX 2: Select only available columns**
available_numeric = [col for col in numeric_features if col in df_clean.columns]
available_categorical = [col for col in categorical_features if col in df_clean.columns]
print(f"Using numeric: {available_numeric}")
print(f"Using categorical: {available_categorical}")

# Prepare X and y
X = df_clean[available_numeric + available_categorical]
y = df_clean['Electric Range']  # Now guaranteed no NaN

# **FIXED Pipeline with imputation**
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline([
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ]), available_numeric),

        ('cat', Pipeline([
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('encoder', OneHotEncoder(drop='first', handle_unknown='ignore'))
        ]), available_categorical)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', Ridge(alpha=1.0))
])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

r2 = r2_score(y_test, y_pred)
print(f"✅ SUCCESS - Test R²: {r2:.3f}")
print(f"Train R²: {r2_score(y_train, pipeline.predict(X_train)):.3f}")
```

```
Original shape: (270262, 16)
NaN in Electric Range: 5
Clean shape (y fixed): (270257, 16)
Using numeric: ['Model Year', 'Legislative District']
Using categorical: ['Make', 'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAFV) Eligibility']
✅ SUCCESS - Test R²: 0.914
Train R²: 0.915
```

In [28]:
```
pip install nbconvert
```

```
Requirement already satisfied: nbconvert in c:\users\lenovo\anaconda3\lib\site-packages (7.16.6)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: beautifulsoup4 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (4.12.3)
Requirement already satisfied: bleach!=5.0.0 in c:\users\lenovo\anaconda3\lib\site-packages (from bleach[css]!=5.0.0->nbconvert) (6.2.0)
Requirement already satisfied: defusedxml in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (3.1.6)
Requirement already satisfied: jupyter-core>=4.7 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (3.1.2)
Requirement already satisfied: nbclient>=0.5.0 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (0.10.2)
Requirement already satisfied: nbformat>=5.7 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (5.10.4)
Requirement already satisfied: packaging in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (24.2)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (2.19.1)
Requirement already satisfied: traitlets>=5.1 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (5.14.3)
Requirement already satisfied: webencodings in c:\users\lenovo\anaconda3\lib\site-packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in c:\users\lenovo\anaconda3\lib\site-packages (from bleach[css]!=5.0.0->nbconvert) (1.4.0)
Requirement already satisfied: platformdirs>=2.5 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-core>=4.7->nbconvert) (4.3.7)
Requirement already satisfied: pywin32>=300 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-core>=4.7->nbconvert) (308)
Requirement already satisfied: jupyter-client>=6.1.12 in c:\users\lenovo\anaconda3\lib\site-packages (from nbclient>=0.5.0->nbconvert) (8.6.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.9.0.post0)
Requirement already satisfied: pyzmq>=23.0 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (26.2.0)
Requirement already satisfied: tornado>=6.2 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.5.1)
Requirement already satisfied: fastjsonschema>=2.15 in c:\users\lenovo\anaconda3\lib\site-packages (from nbformat>=5.7->nbconvert) (2.20.0)
Requirement already satisfied: jsonschema>=2.6 in c:\users\lenovo\anaconda3\lib\site-packages (from nbformat>=5.7->nbconvert) (4.23.0)
Requirement already satisfied: attrs>=22.2.0 in c:\users\lenovo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (24.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\users\lenovo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in c:\users\lenovo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in c:\users\lenovo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.22.3)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (1.17.0)
Requirement already satisfied: soupsieve>1.2 in c:\users\lenovo\anaconda3\lib\site-packages (from beautifulsoup4->nbconvert) (2.5)
```

In [ ]: