```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from ast import literal_eval
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MultiLabelBinarizer
import pickle
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import MultiLabelBinarizer

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)


df = pd.read_csv('/content/carbon emission mitigation 1.xls')

# change display settings to show all columns
pd.set_option('display.max_columns', None)

#rename
# rename columns: replace spaces with underscores
df.columns = df.columns.str.replace(' ', '_')

# convert Gender to Boolean-datatyp
df.rename(columns= {'Sex':'Gender'}, inplace = True)

df.head()
```

```python
print("Full dataset shape is", df.shape)
##############################################

# Berechnet den maximalen Wert einer Spalte, wenn sie numerisch ist
def max_value(column):
```

```
def max_value(column):
    if pd.api.types.is_numeric_dtype(column):  # Überprüfe, ob der Datentyp numerisch ist
        return column.dropna().max() if not column.dropna().empty else np.nan
    return ""

# Gibt die einzigartigen Werte einer Spalte zurück, oder eine Range (falls es eine gibt)
def get_unique_values(column):
    if pd.api.types.is_integer_dtype(column):  # Überprüfe, ob der Datentyp eine Ganzzahl ist
        unique_vals = sorted(set(column.dropna()))
        min_val, max_val = column.min(), column.max()
        if unique_vals == list(range(min_val, max_val + 1)):
            return f"range({min_val},{max_val + 1})"
        return unique_vals
#        return f"between {min_val} and {max_val}"
    return sorted(set(column.dropna()))


def summary(df=df):
    summary_df = pd.DataFrame({
        'data type': df.dtypes.astype(str),
        'missing data': df.isna().sum(),
        'unique values': [get_unique_values(df[col]) for col in df.columns],
        'unique values max': [max_value(df[col]) for col in df.columns],
        'Cardinality': df.nunique()
    })
    return summary_df


# Sortiere nach 'data type' und dann nach 'number of unique values'
summary_df = summary(df).sort_values(by=['data type', 'Cardinality'])
```

Full dataset shape is (151, 20)

```
max_length_col = len(str("'Stove', 'Oven', 'Microwave', 'Grill', 'Airfryer'"))+2
pd.set_option('max_colwidth', max_length_col + 1) #Set the Column Width #You can increase the widt
#pd.set_option('max_colwidth', None) #Set the Column Width #You can increase the width by passing
#pd.reset_option('max_colwidth') #Rückgängig machen

summary_df.loc['Vehicle_Type', 'unique values'] = ', '.join(['diesel', 'electric', 'hybrid', 'lpg'
#summary_df.loc['Vehicle_Type', 'unique values'] = ['diesel', 'electric', 'hybrid', 'lpg', 'petrol

#change values for "Recycling" & "Cooking_With"

for headline in ["Recycling" ,"Cooking_With"]:
    unique_values= set([item for sublist in df[headline].unique() for item in eval(sublist)]) #eva

    summary_df.loc[headline,'unique values'] = str(unique_values)
    summary_df.loc[headline,'Cardinality'] = len(unique_values)


# Setze die maximale Breite einer Spalte auf None, um keine Begrenzung zu haben
#pd.set_option('display.max_colwidth', None)

summary_df
```

```
df.describe()
```

corelations analysis

```
df_corr=df[['CarbonEmission','Vehicle_Monthly_Distance_Km','Transport','Vehicle_Type']].copy()

# Rename 'public' to 'public transport' and the car-typs - to make the information easier to under
df_corr['Vehicle_Type'] = df_corr['Vehicle_Type'].replace({'petrol': 'car (type: petrol)','diesel'
df_corr['Transport'] = df_corr['Transport'].replace({'public': 'public transport', 'private': 'car


##create dummy-variables for correlation metric:
for item in df_corr['Transport'].unique():
    df_corr[str(item)] = df_corr['Transport'].apply(lambda x: 1 if item == x else 0)

unique_vehicle_types = df_corr['Vehicle_Type'].dropna().unique().tolist()
for item in unique_vehicle_types:
    df_corr[str(item)] = df_corr['Vehicle_Type'].apply(lambda x: 1 if item == x else 0)

df_corr.head()
```

```
transport_counts = df_corr['Transport'].value_counts()
labels = [label for label in transport_counts.index]
sizes = transport_counts.values

# Create categories for Vehicle_Monthly_Distance_Km with 10 bins
bins = [0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]
distance_labels = ["0-1,000km", "1,000-2,000km", "2,000-3,000km", "3,000-4,000km", "4,000-5,000km"
                   "5,000-6,000km", "6,000-7,000km", "7,000-8,000km", "8,000-9,000km", "9,000-10,0
df_corr['Distance_Category'] = pd.cut(df_corr['Vehicle_Monthly_Distance_Km'], bins=bins, labels=di

# Calculate the distribution of transport modes within each Distance_Category
counts = df_corr.groupby(['Distance_Category', 'Transport'], observed=True).size().unstack(fill_va

# Create the combined plot
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(16, 7))


# Pie chart
axes[0].pie(sizes, labels=labels, autopct='%1.1f%%', startangle=120)
axes[0].axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
axes[0].set_title("Distribution of Transport Modes")

# Bar plot
counts.plot(kind='bar', ax=axes[1], width=0.8)
axes[1].set_title('Number of People by Transport Mode and Monthly Distance Traveled')
```

```
axes[1].set_xlabel('Monthly Distance Traveled (km)')
axes[1].set_ylabel('Number of People')
axes[1].legend(title='Transport Mode')
plt.xticks(rotation=0)
```

```
plt.figure(figsize=(10, 4))
sns.set(style="whitegrid")  # Set background style to "whitegrid"

ax = sns.boxplot(y='Transport', x='Vehicle_Monthly_Distance_Km', data=df_corr, palette="Set2", ord

# title and axis labels
plt.title('Monthly Distance Traveled by Transport Mode', fontsize=16, weight='bold')
plt.xlabel('Monthly Distance Traveled (km)', fontsize=14)
plt.ylabel('Transport Mode', fontsize=14)


# Remove grid lines
ax.grid(False)
```

```
# Rotate X-axis labels for better readability
plt.xticks(rotation=0, ha='right')

# Adjust layout to prevent overlap
plt.tight_layout()

# save the figure in png-format
plt.savefig('boxplot_transport.png')

plt.show()
```

```
sns.kdeplot(data=df_corr, x="Vehicle_Monthly_Distance_Km", hue="Vehicle_Type")#,common_norm=False)
plt.show()
```

```
correlations = df_corr[['CarbonEmission', 'Vehicle_Monthly_Distance_Km', 'public transport', 'walk

# delete upper diagonal matrix
mask = np.triu(np.ones_like(correlations, dtype=bool))

plt.figure(figsize=(11, 5))  #size of figure
sns.heatmap(correlations,fmt = '.2f', cmap="coolwarm", annot=True, mask=mask,vmax=1,vmin=-1)
plt.show()
```

```
ordinal_variable_order = {
    'Body_Type': ['underweight', 'normal', 'overweight', 'obese'],
    'Diet': ['vegan','vegetarian','pescatarian','omnivore'],
    'How_Often_Shower': ['less frequently','daily', 'twice a day','more frequently'],
    'Social_Activity': ['never', 'sometimes','often'],
    'Frequency_of_Traveling_by_Air': ['never', 'rarely', 'frequently', 'very frequently'],
    'Waste_Bag_Size': ['small','medium', 'large', 'extra large'],
    'Energy_efficiency': ['Yes', 'Sometimes', 'No']
}

# set the ordering
for column, value_ordering in ordinal_variable_order.items():
    df[column] = pd.Categorical(df[column], categories=value_ordering, ordered=True)

#example
df['Waste_Bag_Size'].unique()
```

```
['large', 'extra large', 'small', 'medium']
Categories (4, object): ['small' < 'medium' < 'large' < 'extra large']
```

```
corr_columns =  ['Gender'] + df.select_dtypes(include=[np.number, 'category']).columns.tolist()
df_corr_ordinal = df[corr_columns].copy()
```

```
# convert Gender to Boolean-datatyp
df_corr_ordinal['Gender'] = df['Gender'].map({'male': True, 'female': False})


# encoding for ordinal variables based on defined order
for column, column_ordering in ordinal_variable_order.items():
   mapping = {category: idx for idx, category in enumerate(column_ordering)}
   df_corr_ordinal[column] = df[column].map(mapping)

# delete upper diagonal matrix
mask = np.triu(np.ones_like(df_corr_ordinal.corr(), dtype=bool))

plt.figure(figsize=(21, 7))  #size of figure
#sns.heatmap(df_corr_ordinal.corr(),fmt = '.2f', cmap="coolwarm", annot=True)
sns.heatmap(df_corr_ordinal.corr(),fmt = '.2f', cmap="seismic", annot=True, mask=mask,vmax=1,vmin=
plt.show()
```

```
pd.DataFrame(df_corr_ordinal.corr()["CarbonEmission"].round(3).abs().sort_values(ascending=False)[
```

data preprocessing

```
print("Number of Duplicates:", df.duplicated().sum())
```

```
Number of Duplicates: 0
```

```
df2 = df[['Cooking_With']].copy()
df2['Cooking_With_Grill'] = df2['Cooking_With'].apply(lambda x: 1 if "Grill" in x else 0)
df2['Cooking_With_Airfryer'] = df2['Cooking_With'].apply(lambda x: 1 if "Airfryer" in x else 0)

print("4992 people have both an air fryer and a grill, 5008 people have neither. No one has only o
pd.DataFrame(df2.groupby(["Cooking_With_Airfryer","Cooking_With_Grill"]).size())
```

```
print("unique values:", set([item for sublist in df['Cooking_With'].unique() for item in eval(subl

# Remove "Airfryer" from the 'Cooking_With'-variable
df['Cooking_With'] = df['Cooking_With'].str.replace(", 'Airfryer'", "")

# Check if the removal was successful
print("unique values:", set([item for sublist in df['Cooking_With'].unique() for item in eval(subl
```

```
unique values: {'Microwave', 'Oven', 'Stove', 'Airfryer', 'Grill'}
unique values: {'Microwave', 'Stove', 'Grill', 'Oven'}
```

```
df.isna().sum()
```

```
df_nan = df[["Transport","Vehicle_Type"]].copy()

df_nan['Vehicle_Type'] = df_nan['Vehicle_Type'].fillna('NaN') # to see the NaN in the code below

pd.DataFrame(df_nan.groupby(["Transport","Vehicle_Type"]).size())
```

```
#test: if "Transport"=="public transport" then "Vehicle Type"==NaN
assert df[df["Transport"]=="public"]["Vehicle_Type"].isna().all()  #wenn in der Liste alle True si

#test: if "walk/bicycle" then "Vehicle Type"==NaN
assert df[df["Transport"]=="walk/bicycle"]["Vehicle_Type"].isna().all()  #wenn in der Liste alle T

#test: if "Transport"=="private" then "Vehicle Type"!=NaN
assert not ((df["Transport"]=="private") & (df["Vehicle_Type"].isna())).any() #any weil gibt es ir


df3 = df[['Transport','Vehicle_Type']].copy()

df3['Transport_Vehicle_Type'] = df3['Vehicle_Type'].fillna(df3['Transport'])
df3['car_owner'] = (df3['Transport'] == 'private')  # aufpassen ob 'car' oder 'private' heißt

df3['Vehicle_Type'] = df3['Vehicle_Type'].fillna('NaN') # to see the NaN in the code below

# to see that: 'Transport_Vehicle_Type' & 'car_owner' hold the same information as 'Transport' & '
pd.DataFrame(df3.groupby(['car_owner',"Transport","Vehicle_Type",'Transport_Vehicle_Type']).size()
```

Encoding

```
print("The entries in Recycling are of type:", type(df['Recycling'][3]), "However for encoding we
df['Recycling'][3]
```

```python
def create_dummy_variables_with_mlb(df, column_name):

    # because the data is stored as a string instead of a list
    df[column_name] = df[column_name].apply(eval)

    mlb = MultiLabelBinarizer()
    binarized_data = mlb.fit_transform(df[column_name])
    binarized_df = pd.DataFrame(binarized_data, columns=mlb.classes_)

    df = pd.concat([df, binarized_df], axis=1)
    df = df.drop(columns=column_name)

    return df

df = create_dummy_variables_with_mlb(df, 'Recycling')
df = create_dummy_variables_with_mlb(df, 'Cooking_With')

df.head()
```

```python
numeric_features = ['Monthly_Grocery_Bill', 'Vehicle_Monthly_Distance_Km', 'Waste_Bag_Weekly_Count

nominal_multi_answer_features=['Glass','Metal','Paper','Plastic','Grill','Microwave','Oven','Stove

# Ordinal Variables & Single-Select Nominal Features
categorical_features = ['Body_Type','Diet','How_Often_Shower','Social_Activity','Frequency_of_Trav


all_columns=set(["CarbonEmission",'Vehicle_Type', 'Transport']).union(
    numeric_features,
    categorical_features,
    nominal_multi_answer_features)

assert all_columns  == set(df.columns.tolist())
```

ColumnTransformer

```python
# for 'Vehicle_Type' and 'Transport':
def transport_custom_impute(X):
    # Ersetze NaN in Vehicle_type mit den Werten aus Transport
    X['Transport_Vehicle_Type'] = X['Vehicle_Type'].fillna(X['Transport'])
    return X[['Transport_Vehicle_Type']]
```

```
transport_pipeline = Pipeline(steps=[
    ('transport_imputer', FunctionTransformer(transport_custom_impute, validate=False)),
    ('onehot', OneHotEncoder(drop="first"))
])

preprocessor = ColumnTransformer(transformers=[
        ("numerical", MinMaxScaler(), numeric_features),
        ("transport_vehicletype", transport_pipeline, ['Vehicle_Type', 'Transport']),
        ("categorical", OneHotEncoder(drop="first"), categorical_features)
    ],remainder="passthrough")

X = df.drop(["CarbonEmission"], axis=1)
X_transformed = preprocessor.fit_transform(X)

# To see the ColumnTransformer
preprocessor
```

```
def get_passthrough_columns(column_transformer, X):
    """
    Extracts the columns that are passed through without transformation in the ColumnTransformer.

    Args:
        column_transformer (ColumnTransformer): Fitted ColumnTransformer object.
        X (pd.DataFrame): Original DataFrame before transformation.

    Returns:
        List[str]: List of column names that are passed through.
    """
    passthrough_indices = column_transformer.transformers_[-1][-1]
    return X.columns[passthrough_indices].tolist()


def get_identity_columns(column_transformer, begin_index, end_index):
    """
    Extracts the columns from the ColumnTransformer that remain unchanged in terms of their struct
    meaning they undergo transformations but  the column number remains the same.

    Args:
        column_transformer (ColumnTransformer): Fitted ColumnTransformer object.
        begin_index (int): Starting index of the identity transformers.
        end_index (int): Ending index (exclusive) of the identity transformers.

    Returns:
        List[str]: List of column names that remain unchanged in number.
    """
    col_names = []
    for _, _, col in column_transformer.transformers_[begin_index:end_index]:
```

```
        col_names.extend(col)  # Collect all untransformed column names
    return col_names


def get_onehot_encoded_columns(column_transformer, begin_index, end_index):
    """
    Extracts the OneHotEncoded feature names for the specified transformers in the ColumnTransform

    Args:
        column_transformer (ColumnTransformer): Fitted ColumnTransformer object.
        begin_index (int): Starting index of the ordinal encoders.
        end_index (int): Ending index (exclusive) of the ordinal encoders.

    Returns:
        List[str]: List of one-hot encoded feature names.
    """
    ohe_feature_names = []
    for col_name, _, col_list in column_transformer.transformers_[begin_index:end_index]:
        ohe_features = column_transformer.named_transformers_[col_name].get_feature_names_out(col_
        ohe_feature_names.extend(ohe_features)
    return ohe_feature_names


# Spaltenanzahl bleibt gleich
# name of the first transformers (transformers index 0 till excluding index 1)
numeric_features = get_identity_columns(preprocessor, 0, 1)

# ❌ hier fehlt noch Spaltennnamen für OneHot-Encoding für Transport_Vehicle_Type❌
# Transport-Pipeline Features
transport_encoder = preprocessor.named_transformers_['transport_vehicletype'].named_steps['onehot'
transport_feature_names = transport_encoder.get_feature_names_out(['Transport_Vehicle_Type']).toli

# Spaltenanzahl erhöht
# name of the transformers index 1 till excluding index -1 (excluding passthrough) ❌habe von 1 au
dummy_categorical_features = get_onehot_encoded_columns(preprocessor, 2, -1)

# name of the first three transformers index -1
passthrough_columns = get_passthrough_columns(preprocessor, X)

transformed_feature_names = numeric_features + transport_feature_names + dummy_categorical_feature

X_transformed = pd.DataFrame(X_transformed, columns=transformed_feature_names)


X_transformed.head()
```

```
df.head()
```

Model Training

```
model = LinearRegression()

x = df.drop(["CarbonEmission"],axis=1)
y = df["CarbonEmission"]

X_train, X_test, y_train, y_test = train_test_split(X_transformed, y, train_size = 0.9)

model.fit(X_train, y_train)
```

```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

```
pd.DataFrame({
    'R-squared':[
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)],

    'Mean Absolute Error(MAE)':[
        round(mean_absolute_error(y_train, y_train_pred), 2), # Use the round function here
        round(mean_absolute_error(y_test, y_test_pred), 2)], # Use the round function here

    'Mean Squared Error(MSE)':[
        round(mean_squared_error(y_train, y_train_pred), 2), # Use the round function here
        round(mean_squared_error(y_test, y_test_pred), 2)], # Use the round function here

    'Root Mean Square Error(RMSE)':[
        round(np.sqrt(mean_squared_error(y_train, y_train_pred)), 2), # Use the round function her
        round(np.sqrt(mean_squared_error(y_test, y_test_pred)), 2)] # Use the round function here
    }, index=['Training Set Evaluation','Test Set Evaluation'])
```

```
with open('linear_regression_model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)
```

Prediction

```
X_pred = pd.DataFrame([
  ["overweight","female","pescatarian","daily","coal","walk/bicycle", np.nan,"often",230,"frequent
  ["obese","female","vegetarian","less frequently","natural gas","walk/bicycle", np.nan,"often",11
], columns=[ 'Body_Type', 'Gender', 'Diet', 'How_Often_Shower', 'Heating_Energy_Source',  'Transpo

y_pred = model.predict(preprocessor.transform(X_pred)).round(0)
print("predicted CarbonEmission: ", y_pred)

X_pred
```

```
X_pred= df.loc[0:1, X.columns]
y_pred = model.predict(preprocessor.transform(X_pred)).round(0)
print("predicted CarbonEmission of first 2 persons: ", y_pred)

y_true = list(df.loc[0:1, 'CarbonEmission'])
print("actual CarbonEmission of first 2 persons: ", y_true)

X_pred
```

```
!pip install streamlit
```

```
Requirement already satisfied: streamlit in /usr/local/lib/python3.10/dist-packages (1.41.1)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: blinker<2,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.10/dist-packages (f
```

```
Requirement already satisfied: protobuf<6,>=3.20 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: rich<14,>=10.14.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (fror
Requirement already satisfied: typing-extensions<5,>=4.3.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: watchdog<7,>=2.1.5 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pydeck<1,>=0.8.0b4 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from altair<
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/dist-packages (fror
```