Predicting CO2 Emission Per Capita for a country using energy consumptions

by Tony Shum, Jing Wen, Aishwarya Nadimpally, Weilin Han

In [1]:
```
# Initialize packages
import pandas as pd
import altair as alt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from scipy.stats import randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer,mean_squared_error,r2_score
import matplotlib.pyplot as plt
```

Summary

Here we attempt to build a prediction model using the k-nearest neighbours algorithm which can use energy consumption and energy generation measurements to predict CO2 emission of certain country of next year. Our final prediction model perform pretty well on unseen test dataset, with R^2 of 0.975 and an overall accuracy calculated to be 0.976. However, the model predict CO2 emission by finding the existing cases in the training data set which is most similiar to unseen data, thus, if there is a case in unseen data set of which measurements are beyond the ranges in training data set (ie. massive increase of energy usage or energy efficiency increase or new type of clean energy), then the prediciton might not be accurate, thus we recomment continuing study to improve this prediction model.

Introduction

According to the intergovernmental panel on climate change (IPCC), CO2 emissions are a leading contributor to global warming and climate change (IPCC, 2014). Understanding the correlation between consumption of different types of energy and CO2 emission is critical for formulating policies aimed at reducing emissions and mitigating climate change impacts (IPCC, 2018).

Our project aims to estimate a machine learning model to use energy consumptions per capita to predict CO2 Emission per capita of a country. Our model can be a powerful tool for raising public awareness of the impact of energy consumption on CO2 emission and international agreements on emission reductions. We are hoping that our findings will encourage sustainable behavior, such as reducing energy consumption or opting for green energy alternatives (International Energy Agency, 2018).

Methods

Data

The data set that was used in this project is from World Bank via GAPMINDER.ORG, which is an independent Swedish foundation with no political, religious or economic affiliations and the link can be found https://www.gapminder.org/

Credential

Analysis

Data was split with 80% partitioned as training data and 20% as test data. For model building, we have chosen KNeighborsRegressor (KNN) from DummyRegressor,Ridge, SVR, as we have the highest R^2 score for KNN. The hyperparameter K was chosen using 10-fold cross validation with R^2 as the regression metric.

Results & Discussions

Our prediction model performed well on test data, with a final overall R^2 of 0.976, which is promising for predicting a country's CO2 emission per capita given the energy generation and consumption data. Our model has small deviation from residual to the ground truth, as we have RMSE of 1.34 meaning that our model is relatively accurate in terms of CO2 emission prediction.

In [2]:
```
# read datasets and melt dataframe from wide table to long table
# co2 emission per capita
co2_e = pd.read_csv("../data/raw/co2_emissions_tonnes_per_person.csv").melt(
# coal consumption per capita
coal_c = pd.read_csv("../data/raw/coal_consumption_per_cap.csv").melt(id_var
# electricity generation per capita
```

```
elec_g = pd.read_csv("../data/raw/electricity_generation_per_person.csv").me
# electricity consumption per capita
elec_c = pd.read_csv("../data/raw/electricity_use_per_person.csv").melt(id_v
# hydro generation per capita
hydro_g = pd.read_csv("../data/raw/hydro_power_generation_per_person.csv").m
# nuclear generation per capita
nuclear_g = pd.read_csv("../data/raw/nuclear_power_generation_per_person.csv
# gas generation per capita
gas_g = pd.read_csv("../data/raw/natural_gas_production_per_person.csv").mel
# oil consumption per capita
oil_c = pd.read_csv("../data/raw/oil_consumption_per_cap.csv").melt(id_vars=
# oil generation per capita
oil_g = pd.read_csv("../data/raw/oil_production_per_person.csv").melt(id_var
```

In [3]:

```
#merging data
dataframes = [co2_e, coal_c, elec_g, elec_c, hydro_g, nuclear_g, gas_g, oil_
merged_df = co2_e
for df in dataframes[1:]:
    # Merge each DataFrame with the merged DataFrame
    merged_df = pd.merge(merged_df, df, on=['country','year'], how ='outer')
merged_df
```

Out[3]:

	country	year	co2_e	coal_c	elec_g	elec_c	hydro_g	nuclear_g	gas_g
0	Afghanistan	1800	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Angola	1800	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Albania	1800	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Andorra	1800	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	UAE	1800	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
42803	Yemen	2019	NaN	NaN	NaN	NaN	NaN	NaN	0.017
42804	Zambia	2019	NaN	NaN	NaN	NaN	NaN	NaN	NaN
42805	Zimbabwe	2019	NaN	NaN	NaN	NaN	NaN	NaN	NaN
42806	Equatorial Guinea	2019	NaN	NaN	NaN	NaN	NaN	NaN	NaN
42807	Chad	2019	NaN	NaN	NaN	NaN	NaN	NaN	NaN

42808 rows × 11 columns

In [4]:

```
#EDA to remove years and countries with too many NaN
cleaned_df = merged_df.query('not co2_e.isna() and not coal_c.isna() and not
# remove rows that co2_e, coal_c, elec_c and oil_c do not have na value. But
# Filling NaN with 0.
cleaned_df = cleaned_df.fillna(0)
cleaned_df = cleaned_df.reset_index().drop(columns='index')
cleaned_df
```

Out[4]:

	country	year	co2_e	coal_c	elec_g	elec_c	hydro_g	nuclear_g	gas_g
0	Australia	1965	10.70	1.54	0	2630	0.0628	0	0.000
1	Austria	1965	5.23	0.696	0	2310	0.186	0	0.000
2	Belgium	1965	11.20	2.03	0	2160	0.00248	0	0.000
3	Canada	1965	12.80	0.788	0	6910	0.516	0.00056	0.000
4	Switzerland	1965	5.22	0.219	0	3590	0.364	0	0.000
...	...	...	...	...	...	...	...	...	...
3340	South Africa	2014	8.86	1.64	4670	4180	0	0	0.000
3341	Indonesia	2015	1.96	0.198	906	910.0	0	0	0.253
3342	Indonesia	2016	2.15	0.204	948	956.0	0	0	0.247
3343	Indonesia	2017	2.21	0.216	962	1020.0	0	0	0.236
3344	Indonesia	2018	2.30	0.253	998	1060.0	0	0	0.234

3345 rows × 11 columns

In [5]:
```
# save dataframe
cleaned_df.to_csv('../data/processed/save_the_earth_processed_data.csv', inc
```

In [6]:
```
# read data
df = pd.read_csv("../data/processed/save_the_earth_processed_data.csv", inde
```

Exploratory Data Analysis (EDA)

From our data, we have no NA or missing data, but need to change the data type of No. 3 column to No. 7 column to float. We also need need to clean up data and unify the units, such change 20u to 20e-6 and 15.1k to 15.1e3.

In [7]:
```
# Column data types and information about dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3345 entries, 0 to 3344
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   country    3345 non-null   object 
 1   year       3345 non-null   int64  
 2   co2_e      3345 non-null   float64
 3   coal_c     3345 non-null   object 
 4   elec_g     3345 non-null   object 
 5   elec_c     3345 non-null   object 
 6   hydro_g    3345 non-null   object 
 7   nuclear_g  3345 non-null   object 
 8   gas_g      3345 non-null   float64
 9   oil_c      3345 non-null   float64
 10  oil_g      3345 non-null   float64
dtypes: float64(4), int64(1), object(6)
memory usage: 313.6+ KB
```

Cleaning up of Data

In [8]:
```
# covert column data type
cols_to_convert = df.columns[3:8]

# unify the units, e.g. ug to g; kg to g
df[cols_to_convert] = df[cols_to_convert].replace(to_replace=r'(\d+)µ', valu
df[cols_to_convert] = df[cols_to_convert].replace(to_replace=r'(\d+(?:\.\d+)

df[cols_to_convert] = df[cols_to_convert].apply(pd.to_numeric)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3345 entries, 0 to 3344
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   country    3345 non-null   object 
 1   year       3345 non-null   int64  
 2   co2_e      3345 non-null   float64
 3   coal_c     3345 non-null   float64
 4   elec_g     3345 non-null   float64
 5   elec_c     3345 non-null   float64
 6   hydro_g    3345 non-null   float64
 7   nuclear_g  3345 non-null   float64
 8   gas_g      3345 non-null   float64
 9   oil_c      3345 non-null   float64
 10  oil_g      3345 non-null   float64
dtypes: float64(9), int64(1), object(1)
memory usage: 313.6+ KB
```

Split the data

In [9]:
```
# split data
train_df, test_df = train_test_split(df, test_size=0.2, random_state=123)
```

```
train_df
```

Out[9]:

	country	year	co2_e	coal_c	elec_g	elec_c	hydro_g	nuclear_g	gas_g
119	Italy	1969	5.08	0.1840	0.0	1930.0	0.06680	0.00273	0.0000
2559	Vietnam	2004	1.06	0.1070	556.0	488.0	0.01860	0.00000	0.0416
353	Australia	1974	12.70	1.6700	0.0	4580.0	0.08290	0.00000	0.3390
1889	Hungary	1996	6.12	0.4150	3400.0	3160.0	0.00173	0.11800	0.0000
48	Portugal	1966	1.34	0.0685	0.0	543.0	0.05120	0.00000	0.0000
...	...	...	...	...	...	...	...	...	...
2154	Romania	1999	4.07	0.3150	2270.0	1940.0	0.07060	0.02010	0.5020
3089	Qatar	2011	39.20	0.0000	15100.0	16000.0	0.00000	0.00000	63.5000
1766	Saudi Arabia	1994	16.90	0.0000	5820.0	4790.0	0.00000	0.00000	1.9200
1122	Slovak Republic	1985	11.50	1.6400	4360.0	4840.0	0.03540	0.15700	0.0000
1346	Spain	1989	5.75	0.4940	3760.0	3440.0	0.04290	0.12400	0.0000

2676 rows × 11 columns

In [10]:
```
train_df.shape
```

Out[10]: (2676, 11)

In [11]:
```
test_df.shape
```

Out[11]: (669, 11)

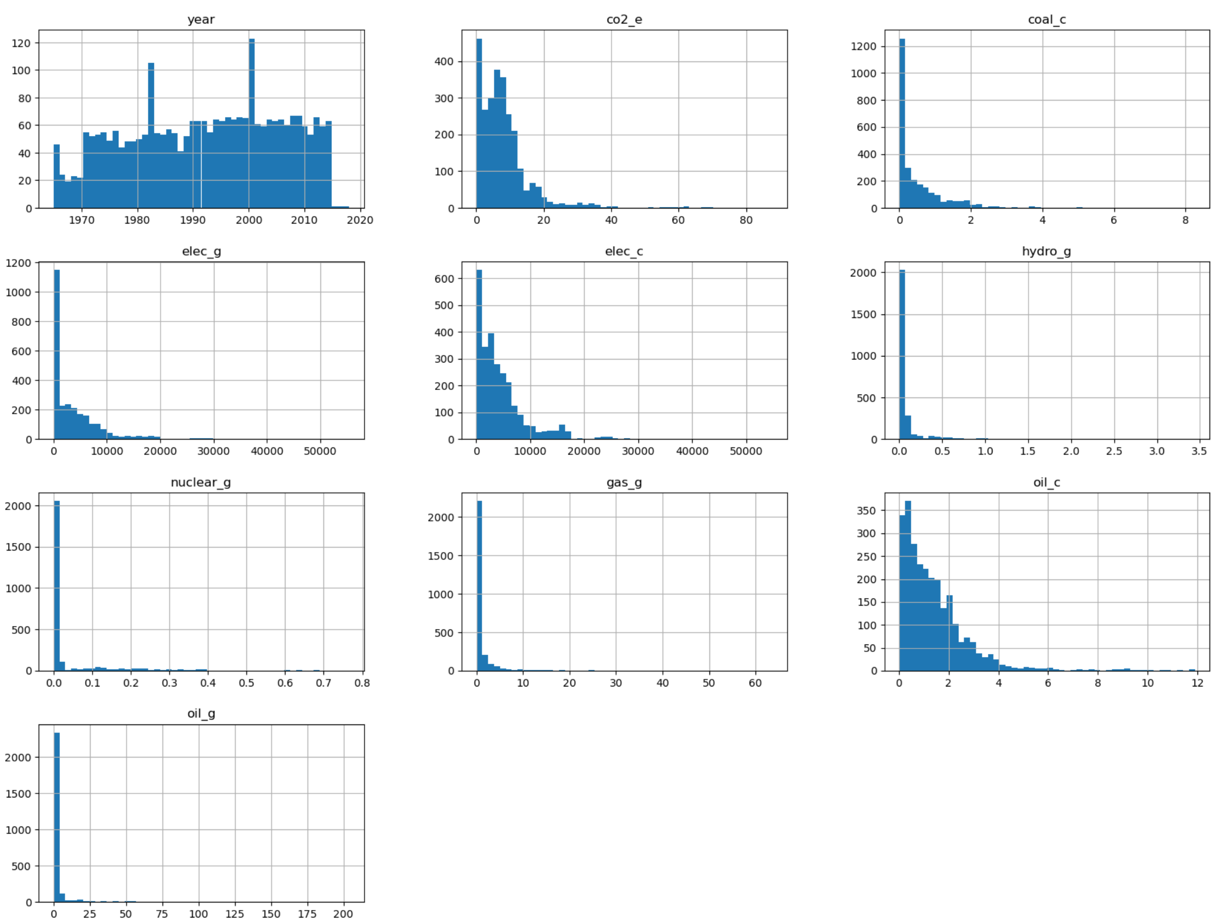
In [12]:
```
# statistical summary for all numeric columns
train_df.describe(include='all')
```

Out[12]:

	country	year	co2_e	coal_c	elec_g	elec
count	2676	2676.000000	2676.000000	2676.000000	2676.000000	2676.0000
unique	78	NaN	NaN	NaN	NaN	Na
top	USA	NaN	NaN	NaN	NaN	Na
freq	44	NaN	NaN	NaN	NaN	Na
mean	NaN	1992.194694	8.407374	0.529499	3855.268834	4655.4883
std	NaN	13.526486	8.757477	0.815835	5660.509816	5257.3670
min	NaN	1965.000000	0.052700	0.000000	0.000000	10.2000
25%	NaN	1981.000000	3.270000	0.018375	0.000000	1220.0000
50%	NaN	1993.000000	6.730000	0.212500	1950.000000	3210.0000
75%	NaN	2004.000000	10.500000	0.741500	5615.000000	6062.5000
max	NaN	2018.000000	87.700000	8.260000	55400.000000	54800.0000

In [13]:

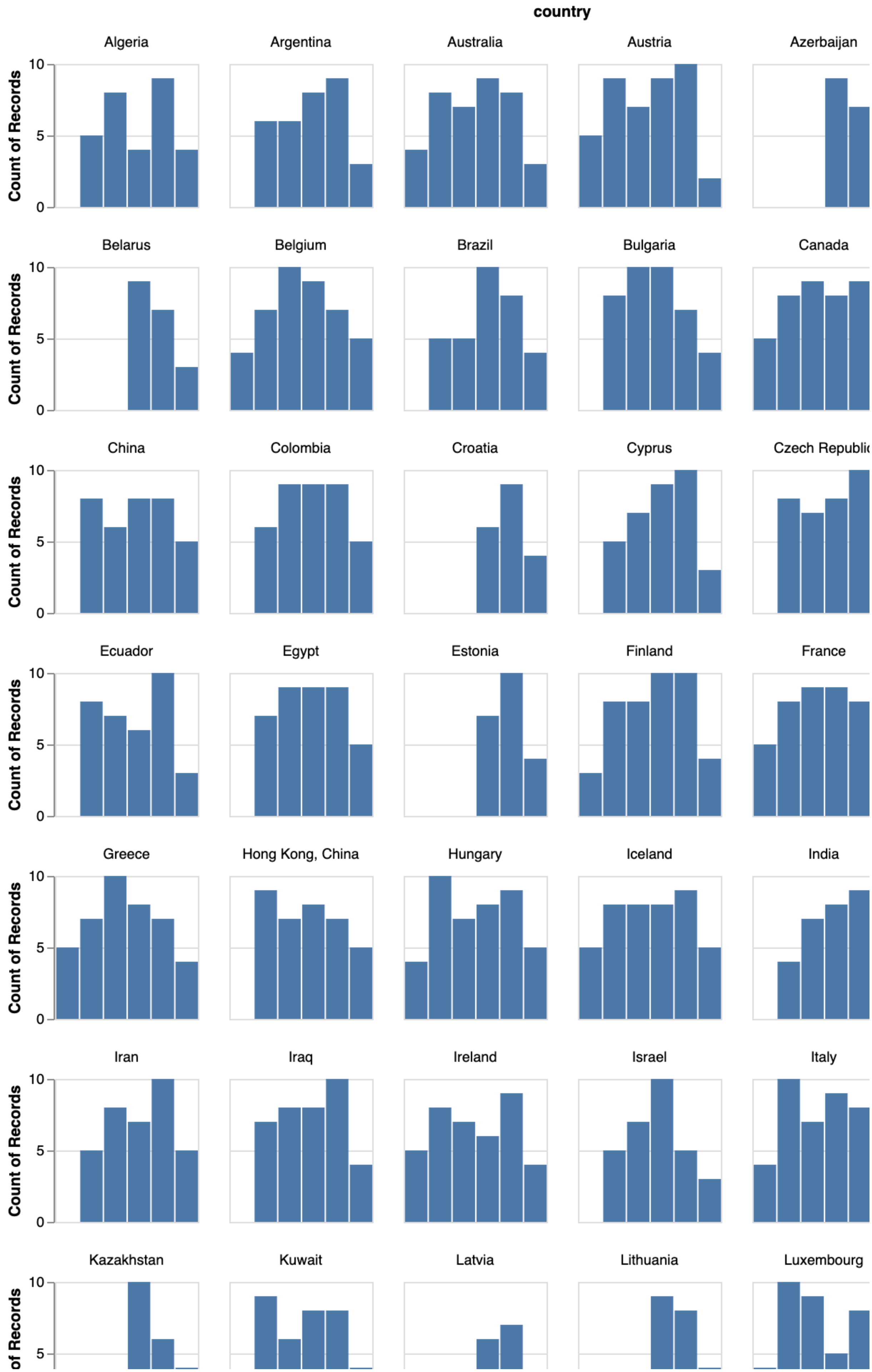
```
# Distributions for all numerical columns
train_df.hist(bins=50, figsize=(20, 15));
```

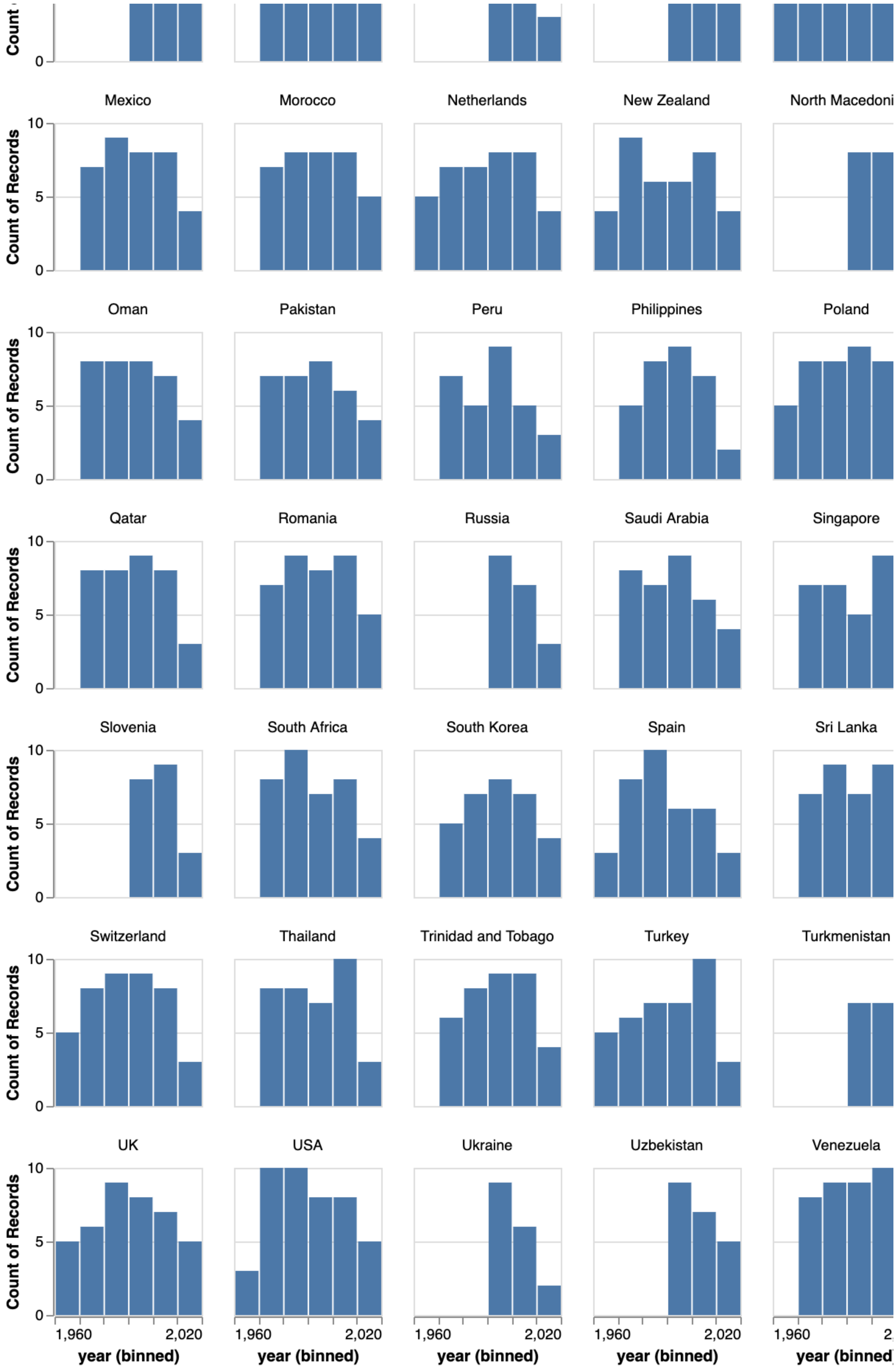


In [14]:

```
# distribution for each country
country_dist = alt.Chart(train_df).mark_bar().encode(
    alt.X('year').bin(maxbins=10),
```

```
    y='count()'
).properties(
    width=100,
    height=100
).facet(
    'country',
    columns=6
)

country_dist.display()
```


country
Algeria
Argentina
Australia
Austria
Azerbaijan
Belarus
Belgium
Brazil
Bulgaria
Canada
China
Colombia
Croatia
Cyprus
Czech Republic
Ecuador
Egypt
Estonia
Finland
France
Greece
Hong Kong, China
Hungary
Iceland
India
Iran
Iraq
Ireland
Israel
Italy
Kazakhstan
Kuwait
Latvia
Lithuania
Luxembourg
Count of Records
0
5
10


Count
Mexico
Morocco
Netherlands
New Zealand
North Macedoni
Oman
Pakistan
Peru
Philippines
Poland
Qatar
Romania
Russia
Saudi Arabia
Singapore
Slovenia
South Africa
South Korea
Spain
Sri Lanka
Switzerland
Thailand
Trinidad and Tobago
Turkey
Turkmenistan
UK
USA
Ukraine
Uzbekistan
Venezuela
Count of Records
0
5
10
1,960
2,020
year (binned)

The Spearmean's rank correlation test below revealed some potential correlations between the following columns: co2_e vs elec_c, co2_e vs oil_c, elec_c vs oil_c, and gas_g vs oil_g.

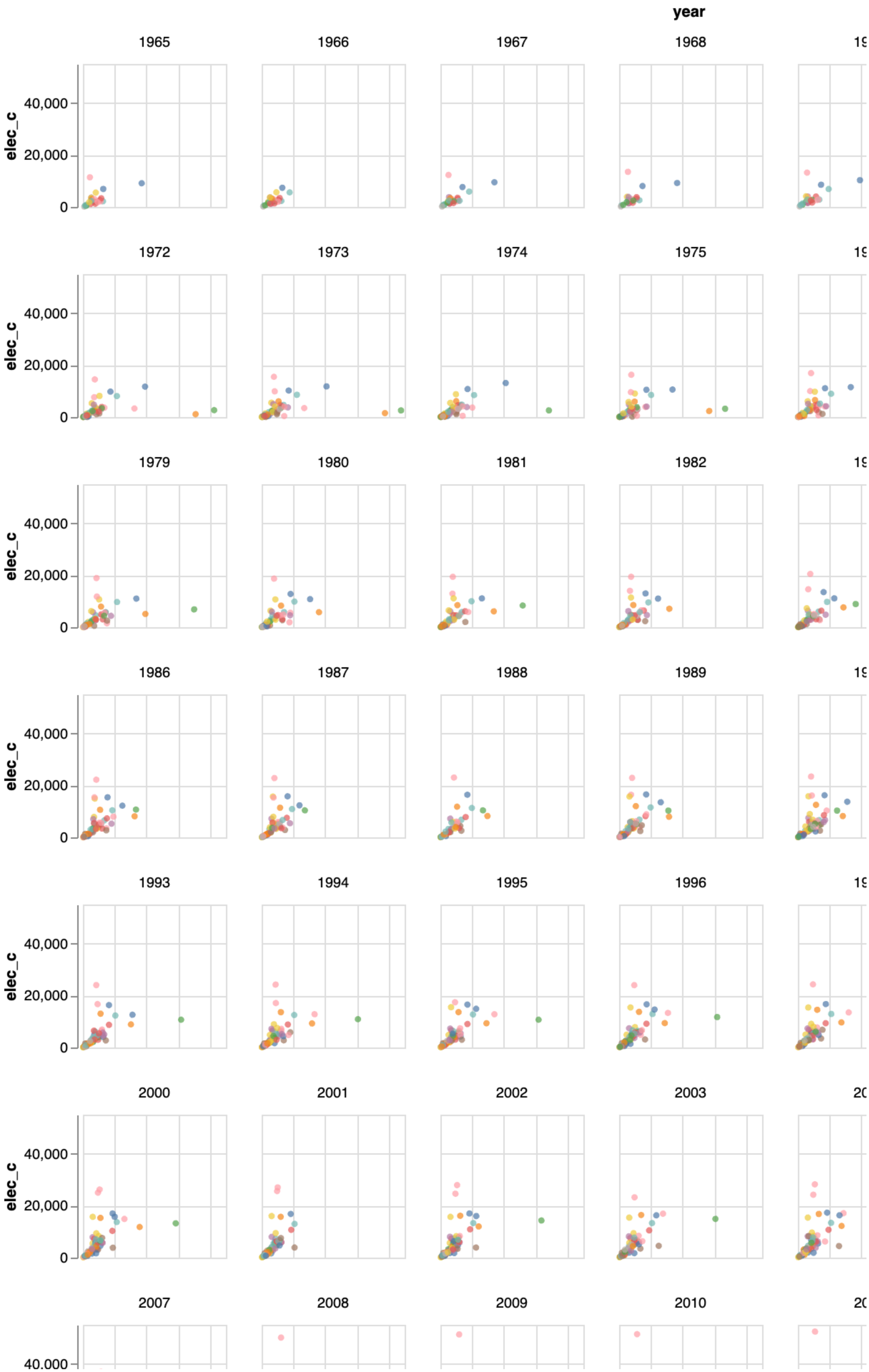
In [15]:
```
# finding potential correlation between numeric columns
num_col = train_df.select_dtypes(include=['int64', 'float64']).columns.tolis

train_df[num_col].corr('spearman').style.background_gradient()
```

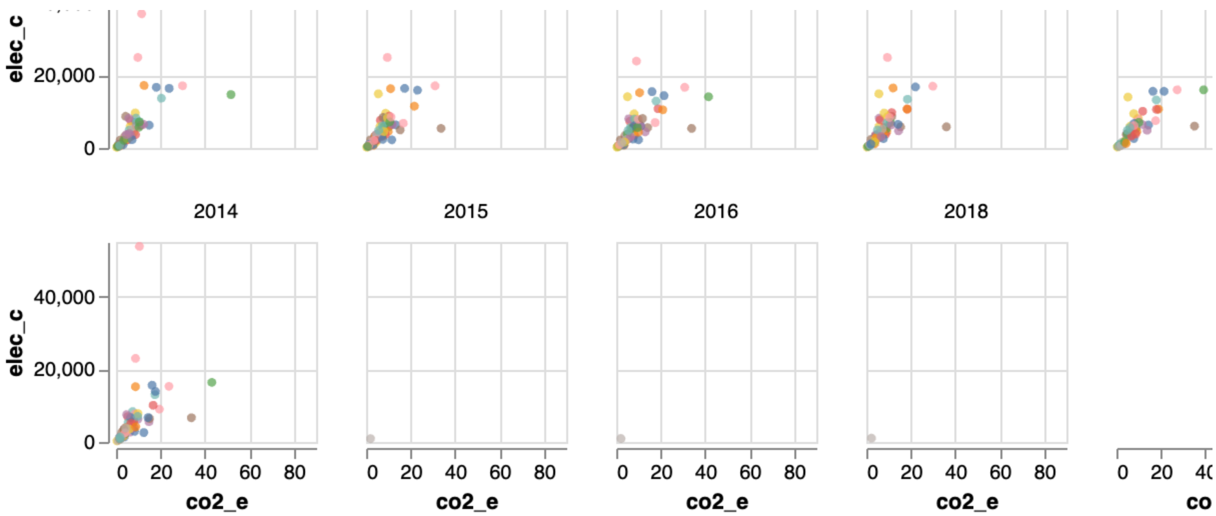
Out[15]:

	year	co2_e	coal_c	elec_g	elec_c	hydro_g	nuclear_
year	1.000000	0.038270	0.003105	0.703007	0.253719	-0.168216	0.00749
co2_e	0.038270	1.000000	0.451953	0.423597	0.820880	-0.028447	0.22798
coal_c	0.003105	0.451953	1.000000	0.222267	0.481075	0.274282	0.39534
elec_g	0.703007	0.423597	0.222267	1.000000	0.641311	0.031603	0.22669
elec_c	0.253719	0.820880	0.481075	0.641311	1.000000	0.220219	0.33632
hydro_g	-0.168216	-0.028447	0.274282	0.031603	0.220219	1.000000	0.34448
nuclear_g	0.007491	0.227988	0.395342	0.226693	0.336324	0.344484	1.00000
gas_g	0.156081	0.204832	-0.264986	0.131033	0.025810	-0.171269	-0.10599
oil_c	-0.028600	0.813402	0.223110	0.363130	0.817716	0.074902	0.20731
oil_g	0.065862	0.135448	-0.399298	0.043299	-0.041407	-0.116532	-0.21245

We further visualized the correlation between columns of interest above in scatter plots. The plots also revealed that we only have one data point for year 2015 to 2018, we can consider exclude these years in the training dataset.

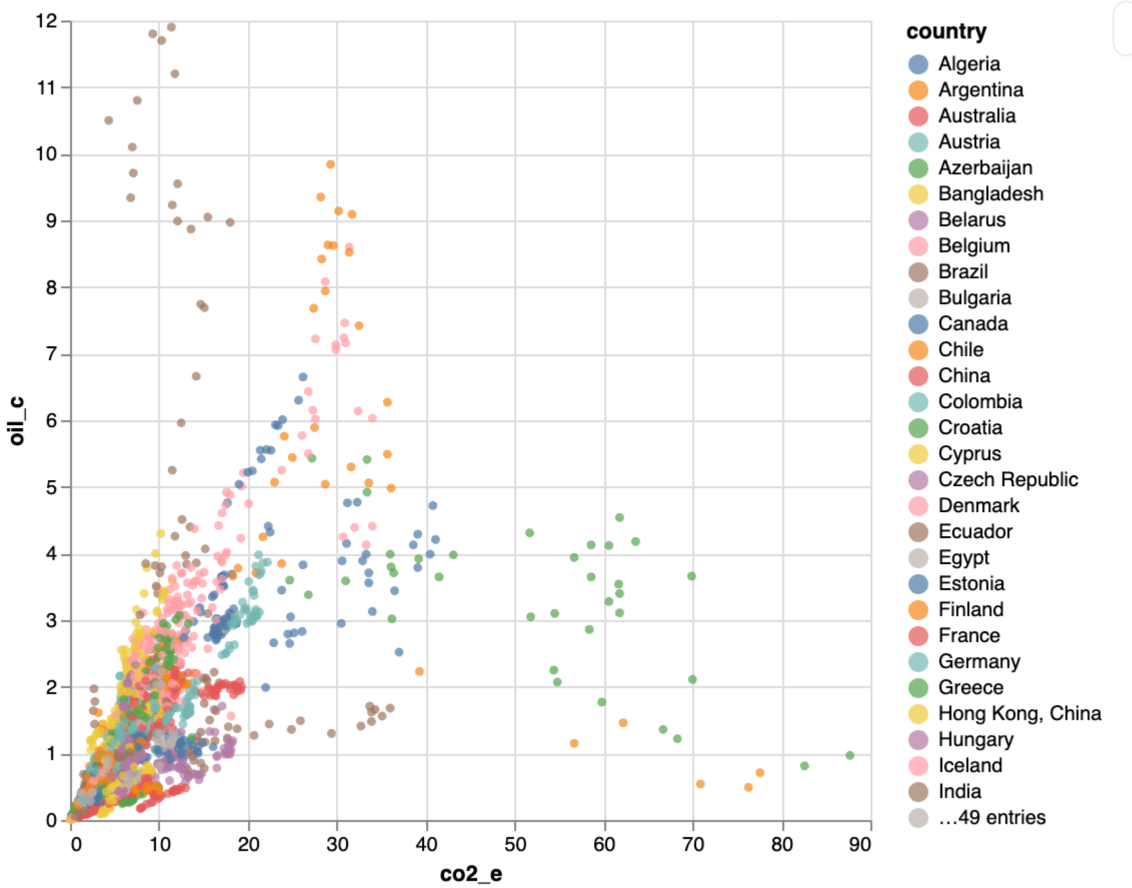
In [16]:
```
# co2_e (co2_emissions_tonnes_per_person) vs elec_e (electricity_use_per_per

chart_1 = alt.Chart(train_df).mark_circle(size=20).encode(
    x='co2_e',
    y='elec_c',
    color='country',
    tooltip=['co2_e', 'elec_c', 'country']
).properties(
    width=100,
    height=100
).facet(
    facet='year',
    columns = 7
)

chart_1.display()
```


year
1965
1966
1967
1968
1972
1973
1974
1975
1979
1980
1981
1982
1986
1987
1988
1989
1993
1994
1995
1996
2000
2001
2002
2003
2007
2008
2009
2010
elec_c
40,000
20,000
0



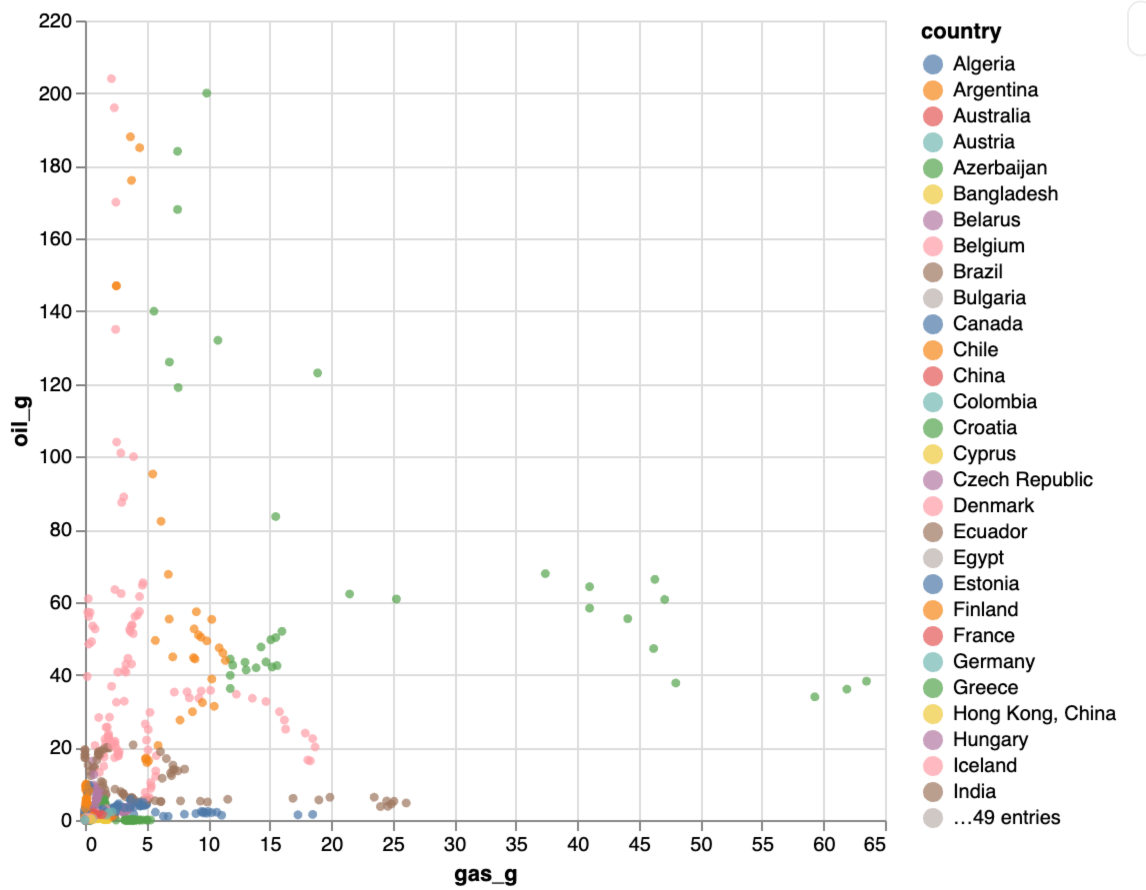
In [17]:

```
# co2_e (co2_emissions_tonnes_per_person) vs oil_c (oil_consumption_per_cap)

chart_2 = alt.Chart(train_df).mark_circle(size=20).encode(
    x='co2_e',
    y='oil_c',
    color='country',
    tooltip=['co2_e', 'oil_c', 'country']
).properties(
    width=400,
    height=400
)

chart_2.display()
```



In [18]:
```
# elec_c (electricity_use_per_person) vs oil_c (oil_consumption_per_cap)

chart_3 = alt.Chart(train_df).mark_circle(size=20).encode(
    x='elec_c',
    y='oil_c',
    color='country',
    tooltip=['elec_c', 'oil_c', 'country']
).properties(
    width=400,
    height=400
)

chart_3.display()
```

In [19]:
```
# gas_g (natural_gas_production_per_person) vs oil_g (oil_production_per_per

chart_4 = alt.Chart(train_df).mark_circle(size=20).encode(
    x='gas_g',
    y='oil_g',
    color='country',
    tooltip=['gas_g', 'oil_g', 'country']
).properties(
    width=400,
    height=400
)

chart_4.display()
```



EDA Conclusion

We have changed the data type to appropriate type and unified the units for each column. We visualized the distribution for all numeric columns and explore potential correlation between columns. We split df into train and test data set (8:2) For pipeline building, it will be beneficial to remove the year 2015 - 2017 because we only have one data point per year.

Export train and test data

In [20]:
```
train_df.to_csv('../data/processed/train_df.csv', index=True)
test_df.to_csv('../data/processed/test_df.csv', index=True)
```

Splitting X and y from train and test data

In [21]:
```
X_train = train_df.drop(columns=["co2_e"])
X_test = test_df.drop(columns=["co2_e"])
y_train = train_df["co2_e"]
y_test = test_df["co2_e"]
```

Preprocessing

Based on the nature of the data and the EDA results, the following assumption and preprocessing would be made

- A **naive assumption** that there is no temporal dependency between observations (i.e. observations among years) is made. `year` would be removed to prevent the model from exploiting the temporal feature for future-looking. Temporal feature treatment, e.g. time series split and time series cross-validation, could be considered later
- Scaling will be applied to all numeric features to standardize them to a common scale.
- OneHotEncoding will be applied to the categorical feature `country`.

In [22]:

```
# Lists of feature names
drop_feats = ['year']
categorical_feats = ['country']
numerical_feats = ['coal_c', 'elec_g', 'elec_c', 'hydro_g', 'nuclear_g', 'ga

# Create the column transformer
preprocessor = make_column_transformer(
    ('drop', drop_feats),
    (StandardScaler(), numerical_feats),
    (OneHotEncoder(handle_unknown='ignore', sparse_output=False, dtype='int'
)

preprocessor.verbose_feature_names_out = False

preprocessor
```

Out[22]:

```
▸ ColumnTransformer
▸ drop    ▸ standardscaler    ▸ onehotencoder
▸ drop    ▸ StandardScaler    ▸ OneHotEncoder
```

Model Training

We used various regression models with R^2 as the scoring metrics and carry out 10-fold cross-validation with each model to find the best performing models. Based on the validation results, the model using k-nearest neighbors (k-nn) algorithm is the best performing model with R^2 of 0.949.

In [24]:

```
models = {
    "Baseline": DummyRegressor(),
    "KNN_reg": KNeighborsRegressor(),
    "Ridge": Ridge(),
    "SVR": SVR(),
}
score_types = {
    "r2": "r2",
}
```

In [25]:
```
cross_val_results = dict()

for name, model in models.items():
    pipe = make_pipeline(preprocessor, model)
    cross_val_results[name] = (
        pd.DataFrame(
            cross_validate(
                pipe,
                X_train,
                y_train,
                cv=10,
                scoring=score_types,
                return_train_score=True,
            )
        )
        .agg(["mean", "std"])
        .round(3)
        .T
    )

cross_val_results_df = pd.concat(
    cross_val_results,
    axis="columns"
)
cross_val_results_df
```

Out[25]:

	Baseline		KNN_reg		Ridge		SVR	
	mean	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
fit_time	0.006	0.002	0.006	0.001	0.010	0.003	0.387	0.053
score_time	0.002	0.001	0.018	0.042	0.004	0.001	0.075	0.019
test_r2	-0.003	0.004	0.953	0.022	0.915	0.021	0.714	0.057
train_r2	0.000	0.000	0.975	0.003	0.926	0.002	0.726	0.006

Hyperparameter Optimization

The hyperparameter `n_neighbors` and `max_categories` was chosen using 10-fold cross validation with R^2 as the classification metric to improve the model performance. Based on the validation results, the KNN model has achieved a R^2(`mean_test_r2`) of 0.975.

In [26]:
```
param_dist = {
    "kneighborsregressor__n_neighbors": randint(1, 20),
    "columntransformer__onehotencoder__max_categories": randint(1, X_train['
}

pipe_best_model = make_pipeline(preprocessor, KNeighborsRegressor())

random_search = RandomizedSearchCV(
    pipe_best_model,
```

```
    param_distributions=param_dist,
    cv=10,
    n_iter=200,
    scoring=score_types,
    n_jobs=-1,
    refit="r2",
    return_train_score=True,
)
random_search.fit(X_train, y_train)
```

Out[26]:

```
▸ RandomizedSearchCV
▸ estimator: Pipeline
  ▸ columntransformer: ColumnTransformer
  ▸ drop      ▸ standardscaler      ▸ onehotencoder
  ▸ drop      ▸ StandardScaler      ▸ OneHotEncoder
        ▸ KNeighborsRegressor
```

In [27]:

```
pd.DataFrame(random_search.cv_results_)[['param_columntransformer__onehotenc
                                         'param_kneighborsregressor__n_neigh
                                         'mean_test_r2',
                                         'std_test_r2']].sort_values('mean_t
```

Out[27]:

	param_columntransformer__onehotencoder__max_categories	param_kneighborsre
93	26	
68	25	
46	24	
67	3	
103	6	
23	5	
153	7	
91	1	
130	23	
156	13	
101	44	
190	28	
64	64	
128	58	
61	52	
72	70	
15	65	
107	32	
75	29	
48	27	

In [28]:

```
# Scaled data export
scaled_X_train = random_search.best_estimator_.named_steps['columntransforme
scaled_X_test = random_search.best_estimator_.named_steps['columntransformer

scaled_X_train = pd.DataFrame(scaled_X_train, columns=random_search.best_est
                              index=X_train.index)
scaled_X_test = pd.DataFrame(scaled_X_test, columns=random_search.best_estim
                             index=X_test.index)

scaled_X_train.to_csv("../data/processed/scaled_save_the_earth_train_data.cs
scaled_X_test.to_csv("../data/processed/scaled_save_the_earth_test_data.csv"
```

In [29]:

```
random_search.best_params_
```

Out[29]:

```
{'columntransformer__onehotencoder__max_categories': 26,
 'kneighborsregressor__n_neighbors': 1}
```

Test Results

In [30]:
```
random_search.score(X_test, y_test)
```
Out[30]: 0.975645926748788

In [31]:
```
# predicting the values for X_test
predicted = random_search.predict(X_test)
actual = pd.DataFrame(y_test)
actual.reset_index(inplace = True, drop = True)
# adding the predicted and actual values to a data frame
result = pd.DataFrame(predicted, columns = ['predicted'])
result['actual'] = actual

#saving the predictions vs actual file
result.to_csv("../data/processed/predictions_vs_actual.csv", index=False)
```

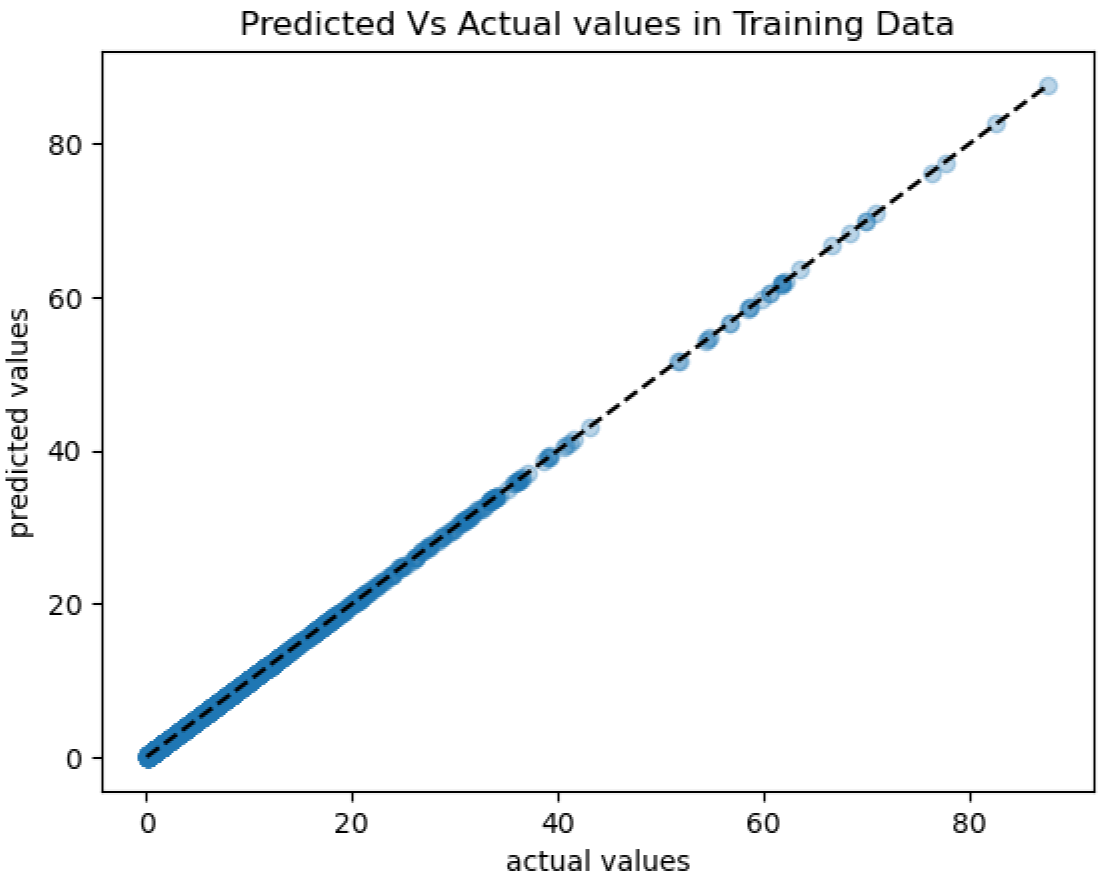
In [32]:
```
# calculating the root mean squared error for test data
np.sqrt(mean_squared_error(actual,predicted))
```
Out[32]: 1.3491335812401586

In [33]:
```
np.sqrt(mean_squared_error(y_train,random_search.predict(X_train)))
```
Out[33]: 0.0

In [34]:
```
#r2 score for training data
r2_score(y_train,random_search.predict(X_train))
```
Out[34]: 1.0

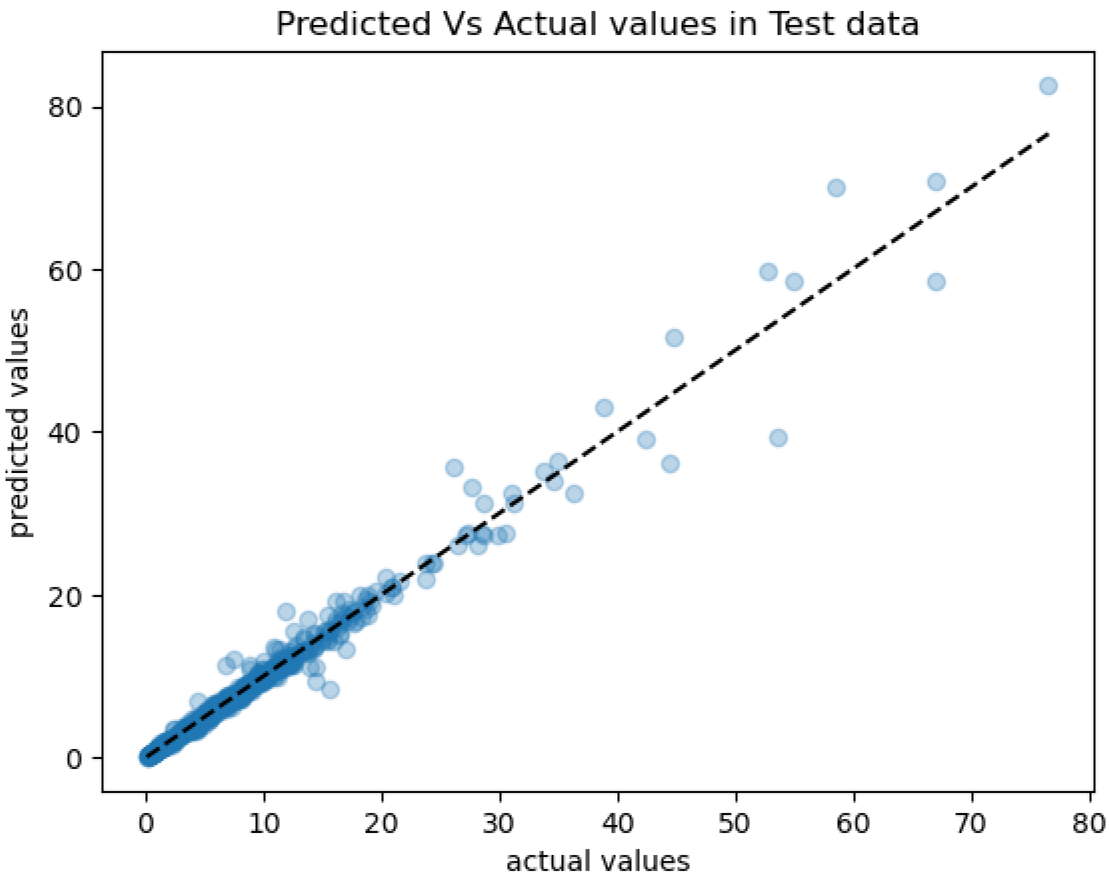
In [35]:
```
#r2 score for test data
r2_score(y_test,random_search.predict(X_test))
```
Out[35]: 0.975645926748788

In [36]:
```
plt.scatter(y_train, random_search.predict(X_train), alpha=0.3)
grid = np.linspace(y_train.min(), y_train.max(), 1000)
plt.plot(grid, grid, "--k")
plt.xlabel("actual values")
plt.ylabel("predicted values");
plt.title("Predicted Vs Actual values in Training Data")
```
Out[36]: Text(0.5, 1.0, 'Predicted Vs Actual values in Training Data')



In [37]:
```
plt.scatter(y_test, random_search.predict(X_test), alpha=0.3)
grid = np.linspace(y_test.min(), y_test.max(), 1000)
plt.plot(grid, grid, "--k")
plt.xlabel("actual values")
plt.ylabel("predicted values");
plt.title("Predicted Vs Actual values in Test data")
```

Out[37]:
```
Text(0.5, 1.0, 'Predicted Vs Actual values in Test data')
```



From the test data plot, we can see that we are under predicting few values. Our model has the accuracy of 97.5% with minimal prediction errors. Our prediction model performed quite well on test data, with a final overall R^2 of 0.976, which is promising for predicting a country's CO2 emission per capita given the energy generation and consumption data. Our model has not less deviation from residual to the ground truth,as we have RMSE of 1.34 which is not too high for our models and it helps for reducing errors.

Limitations and Future Direction

To further improve this model in future with hopes of arriving one that could be used, there are several improvements we can suggest for later revision.As mentioned in Preprocessing, there could possibly be temporal dependency between observations and temporal treatments could be considered. In the EDA above, we discovered there are collinearity between `oil_c` and `elec_c`, `oil_g` and `gas_g`. Though it might not affect the predictive power of models, it harms the interpretation of the coefficients of linear models. Collinearity reduction treatment e.g. feature removal, dimension reduction technique, etc., could be considered. Assumed that co2_emission might be still in increasing trend in the future, KNN may not predict well beyond the range of values input in your training data. Other models with similar predictive power which can predict out-of-range input data could be considered.

References

Morice, C.P., J.J. Kennedy, N.A. Rayner, J.P. Winn, E. Hogan, R.E. Killick, R.J.H. Dunn, T.J. Osborn, P.D. Jones and I.R. Simpson (in press) An updated assessment of near-surface temperature change from 1850: the HadCRUT5 dataset. Journal of Geophysical Research (Atmospheres)

Hannah Ritchie, Max Roser and Pablo Rosado (2020) - "CO_2 and Greenhouse Gas Emissions". Published online at OurWorldInData.org. Retrieved from: 'https://ourworldindata.org/co2-and-greenhouse-gas-emissions'

IPCC, 2014: Climate Change 2014: Synthesis Report. Contribution of Working Groups I, II and III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change. IPCC, Geneva, Switzerland, 151 pp.

IPCC, 2018: Global Warming of 1.5°C. An IPCC Special Report.

International Energy Agency, 2018: World Energy Outlook 2018.