

Homework 2: Climate Change (not done yet)

Homework 2: Climate Change (not done yet)

Question

Preparation

Import data

Exploration and cleaning

Data structure

Statistics

Missing data

Duplication

Outliers

Correlation

Problem 1 — First Model

Read and split

1. Closed form function

2. Formula and R square

3. Significant variables

4. Necessary conditions and application

Problem 2 — Regularization

1. Loss function

2. Compute and regularization

3. Comparison

4. Change λ

Problem 3 — Feature Selection

1. Lesser variables

2. A better model

Problem 4 — Gradient Descent

Define functions

Define parameters

Find theta

Check the result

Question

There have been many studies documenting that the average global temperature has been increasing over the last century. The consequences of a continued rise in global temperature will be dire. Rising sea levels and an increased frequency of extreme weather events will affect billions of people.

In this problem, you will attempt to study the relationship between average global temperature and several other factors. The file `climate_change_1.csv` contains climate data from May 1983 to December 2008. The available variables include: `year`: the observation year.

`month`: the observation month.

`temp`: the difference in degrees Celsius between the average global temperature in that period and a reference value. This data comes from the [Climatic Research Unit at the University of East Anglia](#).

`co2`, `n2o`, `ch4`, `cfc.11`, `cfc.12`: atmospheric concentrations of carbon dioxide (CO_2), nitrous oxide (N_2O), methane (CH_4), trichlorofluoromethane (CCl_3F commonly referred to as $CFC - 11$) and dichlorodifluoromethane (CCl_2F_2 ; commonly referred to as $CFC - 12$), respectively. This data comes from the [ESRL/NOAA Global Monitoring Division](#).

`co2`, `n2o` and `ch4` are expressed in ppmv (parts per million by volume -- i.e., 397 ppmv of CO_2 means that CO_2 constitutes 397 millionths of the total volume of the atmosphere)

`cfc.11` and `cfc.12` are expressed in ppbv (parts per billion by volume).

`aerosols`: the mean stratospheric aerosol optical depth at 550 nm. This variable is linked to volcanoes, as volcanic eruptions result in new particles being added to the atmosphere, which affect how much of the sun's energy is reflected back into space. This data is from the [Godard Institute for Space Studies at NASA](#).

`tsi`: the total solar irradiance (TSI) in W/m^2 (the rate at which the sun's energy is deposited per unit area). Due to sunspots and other solar phenomena, the amount of energy that is given off by the sun varies substantially with time. This data is from the [SOLARIS-HEPPA project website](#).

MEI: multivariate El Nino Southern Oscillation index (MEI), a measure of the strength of the [El Nino/La Nina-Southern Oscillation](#) (a weather effect in the Pacific Ocean that affects global temperatures). This data comes from the [ESRL/NOAA Physical Sciences Division](#).

Preparation

Import data

```
1 import pandas as pd
2 df1 = pd.read_csv('../data/climate_change_1.csv')
3 df2 = pd.read_csv('../data/climate_change_2.csv')
```

Exploration and cleaning

Data structure

```
1 df1.head().round()
```

```
1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }
```

| | Year | Month | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | Temp |
|---|------|-------|-----|-------|--------|-------|--------|--------|--------|----------|------|
| 0 | 1983 | 5 | 3.0 | 346.0 | 1639.0 | 304.0 | 191.0 | 350.0 | 1366.0 | 0.0 | 0.0 |
| 1 | 1983 | 6 | 2.0 | 346.0 | 1634.0 | 304.0 | 192.0 | 352.0 | 1366.0 | 0.0 | 0.0 |
| 2 | 1983 | 7 | 2.0 | 344.0 | 1633.0 | 304.0 | 193.0 | 354.0 | 1366.0 | 0.0 | 0.0 |
| 3 | 1983 | 8 | 1.0 | 342.0 | 1631.0 | 304.0 | 194.0 | 356.0 | 1366.0 | 0.0 | 0.0 |
| 4 | 1983 | 9 | 0.0 | 340.0 | 1648.0 | 304.0 | 194.0 | 357.0 | 1366.0 | 0.0 | 0.0 |

```
1 df2.head().round(2)
```

```
1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }
```

| | Year | Month | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | NO | Temp |
|---|------|-------|------|--------|---------|--------|--------|--------|---------|----------|------|------|
| 0 | 1983 | 5 | 2.56 | 345.96 | 1638.59 | 303.68 | 191.32 | 350.11 | 1366.10 | 0.09 | 2.64 | 0.11 |
| 1 | 1983 | 6 | 2.17 | 345.52 | 1633.71 | 303.75 | 192.06 | 351.85 | 1366.12 | 0.08 | 2.63 | 0.12 |
| 2 | 1983 | 7 | 1.74 | 344.15 | 1633.22 | 303.80 | 192.82 | 353.72 | 1366.28 | 0.07 | 2.63 | 0.14 |
| 3 | 1983 | 8 | 1.13 | 342.25 | 1631.35 | 303.84 | 193.60 | 355.63 | 1366.42 | 0.07 | 2.63 | 0.18 |
| 4 | 1983 | 9 | 0.43 | 340.17 | 1648.40 | 303.90 | 194.39 | 357.46 | 1366.23 | 0.06 | 2.65 | 0.15 |

Statistics

The most significant difference is the variable **NO**. Then explore **basic statistics** with round three:

```
1 df1.describe().round(3)
```

```
1 | .dataframe tbody tr th {
2 |     vertical-align: top;
3 | }
4 |
5 | .dataframe thead th {
6 |     text-align: right;
7 | }
```

| | Year | Month | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | Temp |
|-------|----------|---------|---------|---------|----------|---------|---------|---------|----------|----------|---------|
| count | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 |
| mean | 1995.662 | 6.552 | 0.276 | 363.227 | 1749.825 | 312.392 | 251.973 | 497.525 | 1366.071 | 0.017 | 0.257 |
| std | 7.423 | 3.447 | 0.938 | 12.647 | 46.052 | 5.225 | 20.232 | 57.827 | 0.400 | 0.029 | 0.179 |
| min | 1983.000 | 1.000 | -1.635 | 340.170 | 1629.890 | 303.677 | 191.324 | 350.113 | 1365.426 | 0.002 | -0.282 |
| 25% | 1989.000 | 4.000 | -0.399 | 353.020 | 1722.182 | 308.112 | 246.296 | 472.411 | 1365.717 | 0.003 | 0.122 |
| 50% | 1996.000 | 7.000 | 0.238 | 361.735 | 1764.040 | 311.507 | 258.344 | 528.356 | 1365.981 | 0.006 | 0.248 |
| 75% | 2002.000 | 10.000 | 0.830 | 373.455 | 1786.885 | 316.979 | 267.031 | 540.524 | 1366.363 | 0.013 | 0.407 |
| max | 2008.000 | 12.000 | 3.001 | 388.500 | 1814.180 | 322.182 | 271.494 | 543.813 | 1367.316 | 0.149 | 0.739 |

```
1 | df2.describe().round(3)
```

```
1 | .dataframe tbody tr th {
2 |     vertical-align: top;
3 | }
4 |
5 | .dataframe thead th {
6 |     text-align: right;
7 | }
```

| | Year | Month | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | NC |
|-------|----------|---------|---------|---------|----------|---------|---------|---------|----------|----------|---------|
| count | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 | 308.000 |
| mean | 1995.662 | 6.552 | 0.276 | 363.227 | 1749.825 | 312.392 | 251.973 | 497.525 | 1366.071 | 0.017 | 2.750 |
| std | 7.423 | 3.447 | 0.938 | 12.647 | 46.052 | 5.225 | 20.232 | 57.827 | 0.400 | 0.029 | 0.046 |
| min | 1983.000 | 1.000 | -1.635 | 340.170 | 1629.890 | 303.677 | 191.324 | 350.113 | 1365.426 | 0.002 | 2.630 |
| 25% | 1989.000 | 4.000 | -0.399 | 353.020 | 1722.182 | 308.112 | 246.296 | 472.411 | 1365.717 | 0.003 | 2.722 |
| 50% | 1996.000 | 7.000 | 0.238 | 361.735 | 1764.040 | 311.507 | 258.344 | 528.356 | 1365.981 | 0.006 | 2.764 |
| 75% | 2002.000 | 10.000 | 0.830 | 373.455 | 1786.885 | 316.979 | 267.031 | 540.524 | 1366.363 | 0.013 | 2.787 |
| max | 2008.000 | 12.000 | 3.001 | 388.500 | 1814.180 | 322.182 | 271.494 | 543.813 | 1367.316 | 0.149 | 2.814 |

Missing data

```
1 | df1.info()
```

```
1 | <class 'pandas.core.frame.DataFrame'>
2 | RangeIndex: 308 entries, 0 to 307
3 | Data columns (total 11 columns):
4 | Year          308 non-null int64
5 | Month         308 non-null int64
6 | MEI           308 non-null float64
7 | CO2           308 non-null float64
8 | CH4           308 non-null float64
9 | N2O           308 non-null float64
10 | CFC-11        308 non-null float64
11 | CFC-12        308 non-null float64
12 | TSI           308 non-null float64
13 | Aerosols      308 non-null float64
```

```

14 Temp          308 non-null float64
15 dtypes: float64(9), int64(2)
16 memory usage: 26.6 KB

```

```

1 df2.info()

```

```

1 <class 'pandas.core.frame.DataFrame'>
2 RangeIndex: 308 entries, 0 to 307
3 Data columns (total 12 columns):
4 Year          308 non-null int64
5 Month         308 non-null int64
6 MEI           308 non-null float64
7 CO2           308 non-null float64
8 CH4           308 non-null float64
9 N2O           308 non-null float64
10 CFC-11        308 non-null float64
11 CFC-12        308 non-null float64
12 TSI           308 non-null float64
13 Aerosols      308 non-null float64
14 NO            308 non-null float64
15 Temp          308 non-null float64
16 dtypes: float64(10), int64(2)
17 memory usage: 29.0 KB

```

No missing data were found, then continue.

Duplication

```

1 print('Duplicated rows:', len(df1[df1.duplicated()]), ', then continue.')

```

```

1 Duplicated rows: 0 , then continue.

```

Outliers

Conduct a boxplotting to find out outliers in DF1 and DF2:

```

1 import sys
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 fig1 = plt.figure(figsize=(12,8), dpi=96)
5 for i in range(1, len(df1.columns) + 1):
6     fig1.add_subplot(3, 4, i)
7     df1.iloc[:, [i-1]].boxplot()

```

```

1 range(1, len(df2.columns) + 1)

```

```

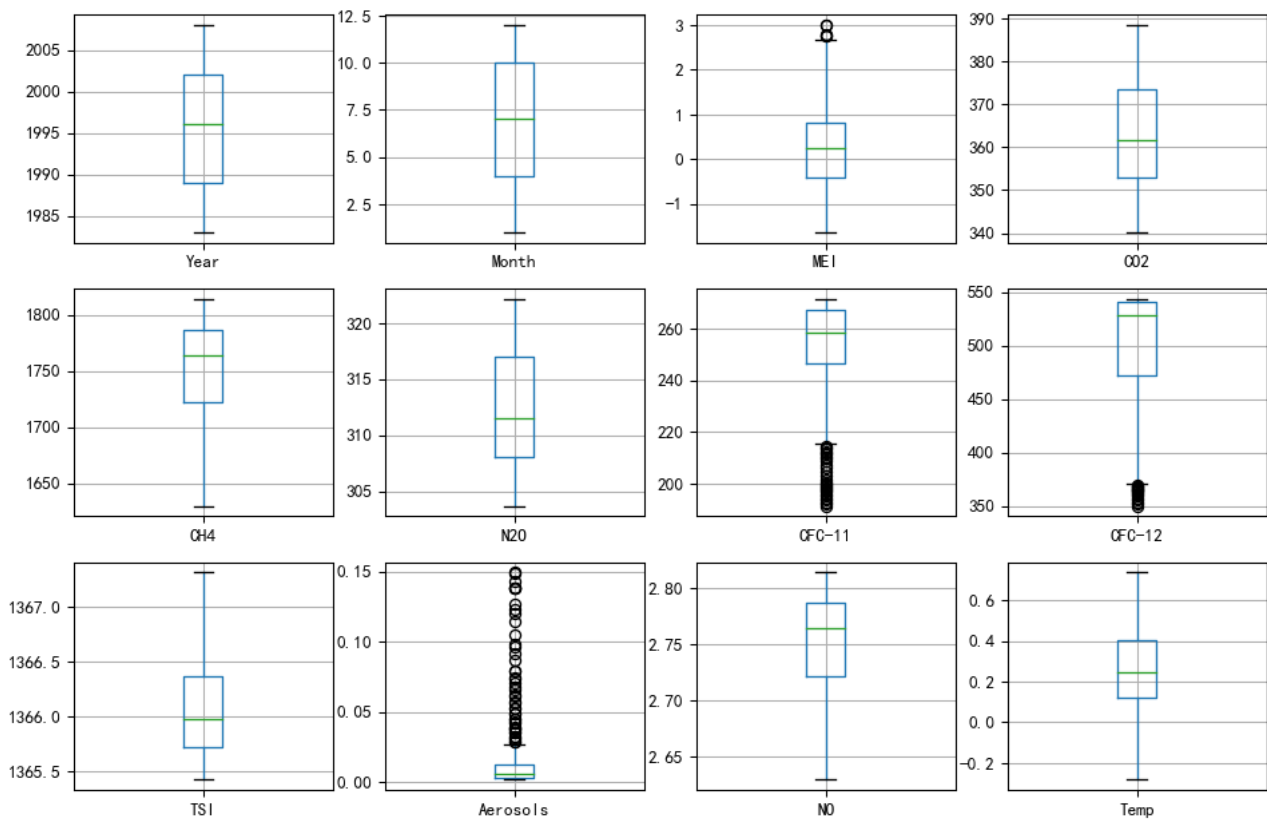
1 range(1, 13)

```

```

1 fig2 = plt.figure(figsize=(12,8), dpi=96)
2 for i in range(1, len(df2.columns) + 1):
3     fig2.add_subplot(3, 4, i)
4     df2.iloc[:, [i-1]].boxplot()

```



Check outliers:

```
1 import ipywidgets as widgets
2 z_slider = widgets.FloatSlider(
3     value=2.9,
4     min=2,
5     max=3.5,
6     step=0.1,
7     description='Threshold:',
8     disabled=False,
9     continuous_update=True,
10    orientation='horizontal',
11    readout=True,
12    readout_format='.1f',
13 )
14 z_slider
```

```
1 FloatSlider(value=2.9, description='Threshold:', max=3.5, min=2.0, readout_format='.1f')
```

```
1 from scipy import stats
2 import numpy as np
3 z = np.abs(stats.zscore(df1['MEI']))
4 outlier_index = np.where(z > z_slider.value)[0]
5 print('Threshold:', z_slider.value)
6 print('Index:', outlier_index)
7 print('Outlier:', [df1['MEI'][i] for i in outlier_index])
```

```
1 Threshold: 2.9
2 Index: [171 172]
3 Outlier: [3.0010000000000003, 3.0]
```

Since rare outliers, ignore at preparation step and continue.

Correlation

Find and plot highly correlated variables ($r > 0.6$ in $df1$, plotting $r > 0.5$):

```

1 corr = df1.corr()
2 high_corr = corr[np.abs(corr) > 0.5].fillna(0)
3 corr[np.abs(corr) > 0.6].fillna('')

```

```

1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }

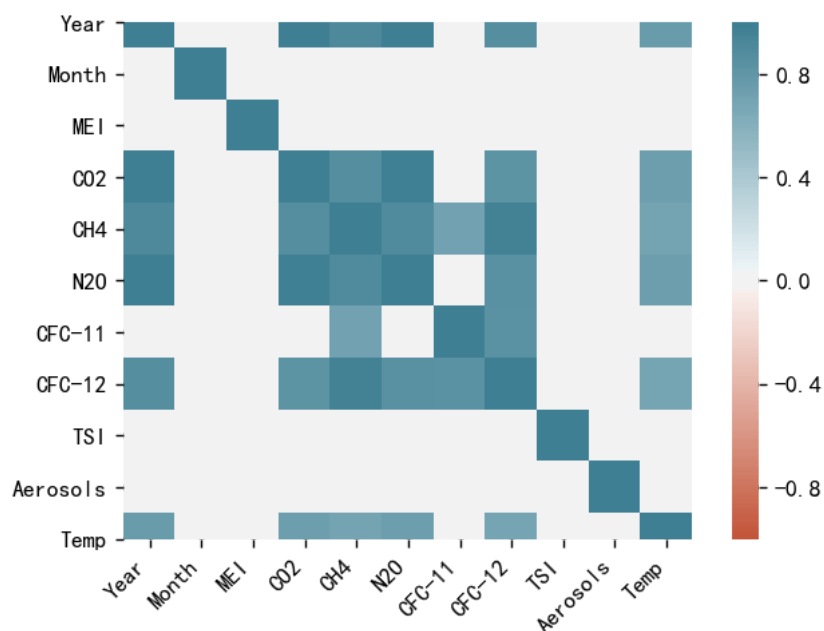
```

| | Year | Month | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | Temp |
|----------|----------|-------|-----|----------|----------|----------|----------|----------|-----|----------|----------|
| Year | 1 | | | 0.985379 | 0.910563 | 0.99485 | | 0.870067 | | | 0.755731 |
| Month | | 1 | | | | | | | | | |
| MEI | | | 1 | | | | | | | | |
| CO2 | 0.985379 | | | 1 | 0.872253 | 0.981135 | | 0.82321 | | | 0.748505 |
| CH4 | 0.910563 | | | 0.872253 | 1 | 0.894409 | 0.713504 | 0.958237 | | | 0.699697 |
| N2O | 0.99485 | | | 0.981135 | 0.894409 | 1 | | 0.839295 | | | 0.743242 |
| CFC-11 | | | | | 0.713504 | | 1 | 0.831381 | | | |
| CFC-12 | 0.870067 | | | 0.82321 | 0.958237 | 0.839295 | 0.831381 | 1 | | | 0.688944 |
| TSI | | | | | | | | | 1 | | |
| Aerosols | | | | | | | | | | 1 | |
| Temp | 0.755731 | | | 0.748505 | 0.699697 | 0.743242 | | 0.688944 | | | 1 |

```

1 plt.figure(dpi=128)
2 ax = sns.heatmap(
3     high_corr,
4     vmin=-1, vmax=1, center=0,
5     cmap=sns.diverging_palette(20, 220, n=200),
6     square=True
7 )
8 ax.set_xticklabels(
9     ax.get_xticklabels(),
10    rotation=45,
11    horizontalalignment='right'
12 );

```



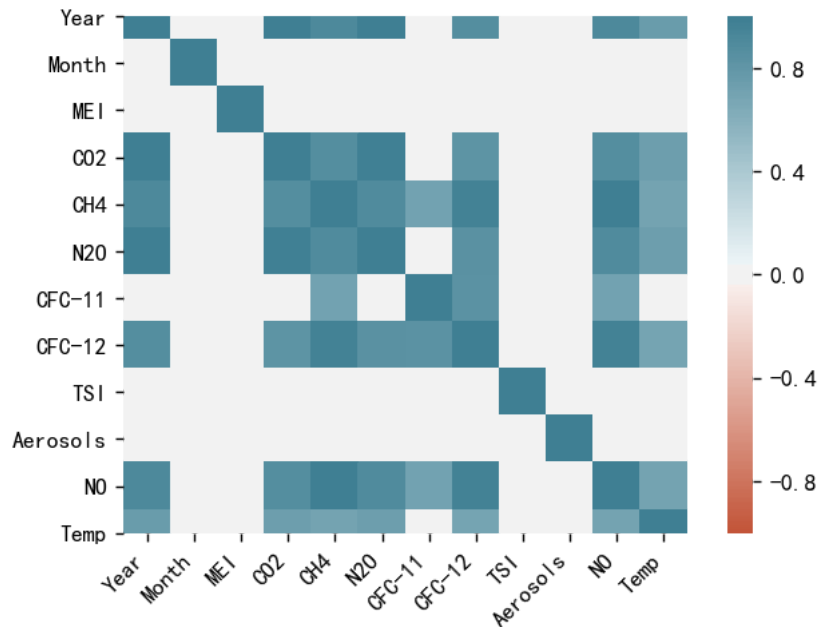
Similarly, correlation in df1:

```
1 corr = df2.corr()
2 high_corr = corr[np.abs(corr) > 0.5].fillna(0)
3 corr[np.abs(corr) > 0.6].fillna('')
```

```
1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }
```

| | Year | Month | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | NO |
|----------|----------|-------|-----|----------|----------|----------|----------|----------|-----|----------|----------|
| Year | 1 | | | 0.985379 | 0.910563 | 0.99485 | | 0.870067 | | | 0.910563 |
| Month | | 1 | | | | | | | | | |
| MEI | | | 1 | | | | | | | | |
| CO2 | 0.985379 | | | 1 | 0.872253 | 0.981135 | | 0.82321 | | | 0.872253 |
| CH4 | 0.910563 | | | 0.872253 | 1 | 0.894409 | 0.713504 | 0.958237 | | | 1 |
| N2O | 0.99485 | | | 0.981135 | 0.894409 | 1 | | 0.839295 | | | 0.894409 |
| CFC-11 | | | | | 0.713504 | | 1 | 0.831381 | | | 0.713504 |
| CFC-12 | 0.870067 | | | 0.82321 | 0.958237 | 0.839295 | 0.831381 | 1 | | | 0.958237 |
| TSI | | | | | | | | | 1 | | |
| Aerosols | | | | | | | | | | 1 | |
| NO | 0.910563 | | | 0.872253 | 1 | 0.894409 | 0.713504 | 0.958237 | | | 1 |
| Temp | 0.755731 | | | 0.748505 | 0.699697 | 0.743242 | | 0.688944 | | | 0.699697 |

```
1 plt.figure(dpi=128)
2 ax = sns.heatmap(
3     high_corr,
4     vmin=-1, vmax=1, center=0,
5     cmap=sns.diverging_palette(20, 220, n=200),
6     square=True
7 )
8 ax.set_xticklabels(
9     ax.get_xticklabels(),
10    rotation=45,
11    horizontalalignment='right'
12 );
```



Potential redundant variables found, however, now the data is prepared for analyzing.

Problem 1 — First Model

We are interested in how changes in these variables affect future temperatures, as well as how well these variables explain temperature changes so far. To do this, first read the dataset `climate_change_1.csv` into Python or Matlab.

Then, split the data into a training set, consisting of all the observations up to and including 2006, and a testing set consisting of the remaining years. A training set refers to the data that will be used to build the model, and a testing set refers to the data we will use to test our predictive ability.

After seeing the problem, your classmate Alice immediately argues that we can apply a linear regression model. Though being a little doubtful, you decide to have a try. To solve the linear regression problem, you recall the linear regression has a closed form solution:

$$\theta = (X^T X)^{-1} X^T Y$$

Read and split

Though data have been prepared in section *Data Preparation*, dataset `df1` has been imported again here following problem description.

Read the dataset:

```
1 # loaded in exploration
2
3 # import pandas as pd
4 # df1 = pd.read_csv('../data/climate_change_1.csv').iloc[:,2:]
```

Split into training set and testing set:

```
1 # Df1 trainset
2 df1_train = df1[df1['Year'] <= 2006].iloc[:,2:]
3 # Check the result
4 df1_train.iloc[[0, 1, -2, -1],:]
```

```
1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }
```


| | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | Temp |
|-----|-------|--------|---------|---------|---------|---------|-----------|----------|-------|
| 0 | 2.556 | 345.96 | 1638.59 | 303.677 | 191.324 | 350.113 | 1366.1024 | 0.0863 | 0.109 |
| 1 | 2.167 | 345.52 | 1633.71 | 303.746 | 192.057 | 351.848 | 1366.1208 | 0.0794 | 0.118 |
| 282 | 1.292 | 380.18 | 1791.91 | 320.321 | 248.605 | 539.500 | 1365.7039 | 0.0049 | 0.440 |
| 283 | 0.951 | 381.79 | 1795.04 | 320.451 | 248.480 | 539.377 | 1365.7087 | 0.0054 | 0.518 |

```

1 # Df1 testet
2 df1_test = df1[df1['Year']>2006].iloc[:,2:]
3 # Check the result
4 df1_test.iloc[[0, 1,-2, -1],:]

```

```

1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }

```

| | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | Temp |
|-----|--------|--------|---------|---------|---------|---------|-----------|----------|-------|
| 284 | 0.974 | 382.93 | 1799.66 | 320.561 | 248.372 | 539.206 | 1365.7173 | 0.0054 | 0.601 |
| 285 | 0.510 | 383.81 | 1803.08 | 320.571 | 248.264 | 538.973 | 1365.7145 | 0.0051 | 0.498 |
| 306 | -0.621 | 384.13 | 1812.37 | 322.013 | 244.225 | 534.906 | 1365.7065 | 0.0048 | 0.394 |
| 307 | -0.666 | 385.56 | 1812.88 | 322.182 | 244.204 | 535.005 | 1365.6926 | 0.0046 | 0.330 |

1. Closed form function

Implement a function `closed_form_1` that computes this closed form solution given the features X , labels y (using Python or Matlab).

Given a pandas `DataFrame`, the features X is the dataframe excluding target y , then:

```

1 import numpy as np # matrix, vector, and linear algebra support
2 from numpy.linalg import inv # matrix inversion
3
4
5 def closed_form_1(X: np.ndarray, y: np.ndarray) -> np.matrix:
6     """
7     To calculate OLS theta(s) given X, y in ndarrays.
8
9     Parameters:
10     -----
11         X: features, IV.
12         y: target variable, DV.
13     Return:
14     -----
15         theta: coefficients
16     """
17
18     X = np.column_stack((np.ones(len(X)), X)) # add x0 = 1 to matrix X
19     theta = inv(X.T @ X) @ X.T @ y
20     #theta = theta[1:].reshape((1,10))
21     return theta
22
23
24 def closed_form_df(df: pd.core.frame.DataFrame, column: int = 8) -> np.matrix:
25     """
26     To calculate OLS theta(s) given data in a DataFrame.
27
28     Parameters:
29     -----
30         df: a DataFrame of data including both IV X and DV y.

```

```

31     column = 8: index number of cloomn where DV y lies. The default value is 8.
32
33     Return:
34     -----
35     theta: coefficients
36     """
37
38     x = df.drop(df.columns[column], axis=1).to_numpy() # X: the features
39     x = np.column_stack((np.ones(len(x)), x)) # add x0 = 1 to matrix X
40     y = df.iloc[:, [column]].to_numpy()
41     ) # y: the results, lower case to emphasize the difference
42     theta = inv(X.T @ X) @ X.T @ y
43     #theta = theta[1:].reshape((1,10))
44     return theta

```

Test `closed_form_1` and `closed_form_df` on `df1`:

```

1 df1_train.drop(df1_train.columns[8], axis=1)

```

```

1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }

```

| | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols |
|-----|-------|--------|---------|---------|---------|---------|-----------|----------|
| 0 | 2.556 | 345.96 | 1638.59 | 303.677 | 191.324 | 350.113 | 1366.1024 | 0.0863 |
| 1 | 2.167 | 345.52 | 1633.71 | 303.746 | 192.057 | 351.848 | 1366.1208 | 0.0794 |
| 2 | 1.741 | 344.15 | 1633.22 | 303.795 | 192.818 | 353.725 | 1366.2850 | 0.0731 |
| 3 | 1.130 | 342.25 | 1631.35 | 303.839 | 193.602 | 355.633 | 1366.4202 | 0.0673 |
| 4 | 0.428 | 340.17 | 1648.40 | 303.901 | 194.392 | 357.465 | 1366.2335 | 0.0619 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 279 | 0.759 | 380.45 | 1762.66 | 319.930 | 248.981 | 539.682 | 1365.7067 | 0.0041 |
| 280 | 0.793 | 378.92 | 1776.04 | 320.010 | 248.775 | 539.566 | 1365.8419 | 0.0043 |
| 281 | 0.892 | 379.16 | 1789.02 | 320.125 | 248.666 | 539.488 | 1365.8270 | 0.0044 |
| 282 | 1.292 | 380.18 | 1791.91 | 320.321 | 248.605 | 539.500 | 1365.7039 | 0.0049 |
| 283 | 0.951 | 381.79 | 1795.04 | 320.451 | 248.480 | 539.377 | 1365.7087 | 0.0054 |

284 rows × 8 columns

```

1 # Given X, and y in numpy arrays
2 x = df1_train.drop(df1_train.columns[8], axis=1).to_numpy() # X: the features
3 y = df1_train.iloc[:, [8]].to_numpy() # y: the results, lower case to emphasize the
   difference
4 X_test = df1_test.drop(df1_train.columns[8], axis=1).to_numpy()
5 y_test = df1_test.iloc[:, [8]].to_numpy()
6 theta = closed_form_1(X, y)
7 theta

```

```

1 array([[ -1.24594260e+02,
2         [ 6.42053134e-02],
3         [ 6.45735927e-03],
4         [ 1.24041896e-04],
5         [-1.65280032e-02],
6         [-6.63048889e-03],
7         [ 3.80810324e-03],
8         [ 9.31410835e-02],
9         [-1.53761324e+00]])

```

```

1 # Given a DataFrame
2 theta = closed_form_df(df1_train).reshape((1,9))
3 theta

```

```

1 [[1.0000000e+00 2.5560000e+00 3.4596000e+02 ... 3.5011300e+02
2     1.3661024e+03 8.6300000e-02]
3 [1.0000000e+00 2.1670000e+00 3.4552000e+02 ... 3.5184800e+02
4     1.3661208e+03 7.9400000e-02]
5 [1.0000000e+00 1.7410000e+00 3.4415000e+02 ... 3.5372500e+02
6     1.3662850e+03 7.3100000e-02]
7 ...
8 [1.0000000e+00 8.9200000e-01 3.7916000e+02 ... 5.3948800e+02
9     1.3658270e+03 4.4000000e-03]
10 [1.0000000e+00 1.2920000e+00 3.8018000e+02 ... 5.3950000e+02
11     1.3657039e+03 4.9000000e-03]
12 [1.0000000e+00 9.5100000e-01 3.8179000e+02 ... 5.3937700e+02
13     1.3657087e+03 5.4000000e-03]]

```

```

1 array([[ -1.24594260e+02,  6.42053134e-02,  6.45735927e-03,
2         1.24041896e-04, -1.65280032e-02, -6.63048889e-03,
3         3.80810324e-03,  9.31410835e-02, -1.53761324e+00]])

```

Using *scipy* to check the result:

```

1 from sklearn.linear_model import LinearRegression as lm
2 l=lm().fit(X, y)
3 l.coef_

```

```

1 array([[ 6.42053134e-02,  6.45735927e-03,  1.24041896e-04,
2         -1.65280033e-02, -6.63048889e-03,  3.80810324e-03,
3         9.31410835e-02, -1.53761324e+00]])

```

Works fine (some differences due to SVD used in *sklearn.LinearRegression*).

2. Formula and R square

Write down the mathematical formula for the linear model and evaluate the model R square on the training set and the testing set.

```

1 df1_train.columns

```

```

1 Index(['MEI', 'CO2', 'CH4', 'N2O', 'CFC-11', 'CFC-12', 'TSI', 'Aerosols',
2       'Temp'],
3       dtype='object')

```

Formula of this model(round(5))

$$\hat{Temp} = -124.594 + 0.06421 * MEI + 0.00646 * CO_2 + 0.00012 * CH_4 - 0.01653 * N_2O - 0.00663 * CFC11 + 0.00381 * CFC12 + 0.00$$

Formula of R-squared

R-squared measures model fitting and can be calculated as: $R^2 = \frac{var(X\hat{\beta})}{var(y)} = \frac{\sum_{i=1}^n (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$

```

1 def predict(x: np.ndarray, theta: np.ndarray) -> np.ndarray:

```

```

2     """
3     To predict y given x and theta.
4
5     Parameters:
6     -----
7
8         x: features, IV.
9         theta: coefficients.
10
11     Return:
12     -----
13         y_hat: predicted value.
14     """
15
16     x = np.column_stack((np.ones(len(x)), x)) # add x0 = 1 to matrix x
17     theta = theta.reshape((1, len(theta)))
18     y_hat = np.sum(x * theta, axis=1)
19     return (y_hat)
20

```

Define a `score` function to calculate R^2 :

```

1 def score(y: np.ndarray, y_hat: np.ndarray) -> float:
2     """
3     To calculate OLS R^2 given data in ndarrays.
4
5     Parameters:
6     -----
7         y: actual labels.
8         y_hat: predicted values.
9
10    Return:
11    -----
12        SST: R^2 caculated based on y and y_hat.
13    """
14
15    mean = y.mean()
16    TSS = np.sum(np.square(y_hat - mean))
17    ESS = np.sum(np.square(y - mean))
18    SST = TSS / ESS
19    return SST

```

On training set:

```

1 x = df1_train.drop(df1_train.columns[8], axis=1).to_numpy()
2 y = df1_train.iloc[:, [8]].to_numpy()
3 rsquare_train = score(y, predict(X, closed_form_1(X, y)))
4 print("R2:", rsquare_train)
5
6 # Use *scipy* to check the result:
7 l=lm().fit(X, y)
8 print("R2 by scipy:", l.score(X, y))

```

```

1 R2: 0.7508932770388234
2 R2 by scipy: 0.7508932770523428

```

On testing set:

```

1 rsquare_test = score(y_test, predict(X_test, closed_form_1(X, y)))
2 print("R2:", rsquare_test)

```

```

1 R2: 0.22517701916248536

```

Works fine.

Evaluation

Based on the formula above, R-squared can be applied in Python to evaluate previous model. On training set: R^2 is 0.75089, while on testing set, R^2 is 0.22518.

*_However, for a multi-variable linear model, $R^2_{adjusted}$ may be a better indicator because the original R^2 is sensitive to the number of features._

3. Significant variables

Which variables are significant in the model?

```
1 import statsmodels.api as sm
2
3 # set an alpha
4 alpha = 0.05
5
6 x2 = sm.add_constant(X)
7 l = sm.OLS(y, x2).fit()
8 pvalues = l.summary2().tables[1]['P>|t|']
9 labels = ['x0: constant'] + ["x" + str(i+1) + ": " + df1_train.columns[i] for i in
10 range(len(df1_train.columns)-1)]
11 variables = pd.DataFrame(np.concatenate([pd.DataFrame(labels), pd.DataFrame(pvalues)], axis=1))
12 variables.columns = ['variable', 'pvalues']
13
14 # print significant variables
15 variables[variables.pvalues < alpha]
```

```
1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }
```

| | Variable | pvalues |
|---|--------------|-------------|
| 0 | x0: constant | 1.43105e-09 |
| 1 | x1: MEI | 4.89889e-20 |
| 2 | x2: CO2 | 0.00505252 |
| 5 | x5: CFC-11 | 5.95729e-05 |
| 6 | x6: CFC-12 | 0.00020972 |
| 7 | x7: TSI | 1.09594e-09 |
| 8 | x8: Aerosols | 5.41127e-12 |

That's to say, significant(alpha=0.05) variables are:

```
1 [i for i in variables[variables.pvalues < alpha].variable.to_numpy()]
```

```
1 ['x0: constant',
2  'x1: MEI',
3  'x2: CO2',
4  'x5: CFC-11',
5  'x6: CFC-12',
6  'x7: TSI',
7  'x8: Aerosols']
```

4. Necessary conditions and application

Write down the necessary conditions for using the closed form solution. And you can apply it to the dataset `climate_change_2.csv`, explain the solution is unreasonable.

Necessary conditions

$X^T X$ must be invertible.

```
1 df2.head(2)
```

```
1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }
```

| | Year | Month | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | NO | Temp |
|---|------|-------|-------|--------|---------|---------|---------|---------|-----------|----------|---------|-------|
| 0 | 1983 | 5 | 2.556 | 345.96 | 1638.59 | 303.677 | 191.324 | 350.113 | 1366.1024 | 0.0863 | 2.63859 | 0.109 |
| 1 | 1983 | 6 | 2.167 | 345.52 | 1633.71 | 303.746 | 192.057 | 351.848 | 1366.1208 | 0.0794 | 2.63371 | 0.118 |

```
1 # Df2 trainset
2 df2_train = df2[df2['Year']<=2006].iloc[:,2:]
3 # Check the result
4 df2_train.iloc[[0, 1,-2, -1],:]
```

```
1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }
```

| | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | NO | Temp |
|-----|-------|--------|---------|---------|---------|---------|-----------|----------|---------|-------|
| 0 | 2.556 | 345.96 | 1638.59 | 303.677 | 191.324 | 350.113 | 1366.1024 | 0.0863 | 2.63859 | 0.109 |
| 1 | 2.167 | 345.52 | 1633.71 | 303.746 | 192.057 | 351.848 | 1366.1208 | 0.0794 | 2.63371 | 0.118 |
| 282 | 1.292 | 380.18 | 1791.91 | 320.321 | 248.605 | 539.500 | 1365.7039 | 0.0049 | 2.79191 | 0.440 |
| 283 | 0.951 | 381.79 | 1795.04 | 320.451 | 248.480 | 539.377 | 1365.7087 | 0.0054 | 2.79504 | 0.518 |

```
1 # Df2 testet
2 df2_test = df2[df2['Year']>2006].iloc[:,2:]
3 # Check the result
4 df2_test.iloc[[0, 1,-2, -1],:]
```

```
1 .dataframe tbody tr th {
2     vertical-align: top;
3 }
4
5 .dataframe thead th {
6     text-align: right;
7 }
```

| | MEI | CO2 | CH4 | N2O | CFC-11 | CFC-12 | TSI | Aerosols | NO | Temp |
|-----|--------|--------|---------|---------|---------|---------|-----------|----------|---------|-------|
| 284 | 0.974 | 382.93 | 1799.66 | 320.561 | 248.372 | 539.206 | 1365.7173 | 0.0054 | 2.79966 | 0.601 |
| 285 | 0.510 | 383.81 | 1803.08 | 320.571 | 248.264 | 538.973 | 1365.7145 | 0.0051 | 2.80308 | 0.498 |
| 306 | -0.621 | 384.13 | 1812.37 | 322.013 | 244.225 | 534.906 | 1365.7065 | 0.0048 | 2.81237 | 0.394 |
| 307 | -0.666 | 385.56 | 1812.88 | 322.182 | 244.204 | 535.005 | 1365.6926 | 0.0046 | 2.81288 | 0.330 |

```

1 # Given X, and y in numpy arrays
2 x_2 = df2_train.drop(df2_train.columns[9], axis=1).to_numpy() # x: the features
3 y_2 = df2_train.iloc[:, [9]].to_numpy() # y: the results, lower case to emphasize the
  difference
4 x_2_test = df2_test.drop(df2_test.columns[9], axis=1).to_numpy()
5 y_2_test = df2_test.iloc[:, [9]].to_numpy()
6 theta = closed_form_1(x_2, y_2)
7 theta

```

```

1 array([[ -1.18459383e+02,
2         [ 6.41762745e-02,
3         [ 6.48209178e-03,
4         [ 6.24389931e-03,
5         [-1.65280032e-02,
6         [-6.63048889e-03,
7         [ 3.80810324e-03,
8         [ 9.31410835e-02,
9         [-1.53761324e+00,
10        [-6.12593018e+00]])

```

Why unreasonable:

Because $X^T X$ is non-invertible.

According to [Andrew NG](#),

When implementing the normal equation in octave we want to use the `pinv` function rather than `inv`. The 'pinv' function will give you a value of θ even if $X^T X$ is not invertible.

If $X^T X$ is noninvertible, the common causes might be having :

- **Redundant features**, where two features are very closely related (i.e. they are linearly dependent)
- **Too many features** (e.g. $m \leq n$). In this case, delete some features or use "regularization" (to be explained in a later lesson).

Solutions to the above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.

In this case, many variables (as mentioned in the first section exploration) are highly correlated.

Problem 2 — Regularization

Regularization is a method to boost robustness of model, including L1 regularization and L2 regularization.

1. Loss function

Please write down the loss function for linear model with L1 regularization, L2 regularization, respectively.

L1, Lasso Regression: $J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j| \right]$

L2, Ridge Regression: $J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$

2. Compute and regularization

The closed form solution for linear model with L2 regularization: $\theta = (X^T X + \lambda I)^{-1} X^T Y$ where I is the identity matrix. Write a function `closed_form_2` that computes this closed form solution given the features X , labels Y and the regularization parameter λ (using Python or Matlab).

```

1 def closed_form_2(X: np.ndarray, y: np.ndarray, lambda: float) -> np.ndarray:
2     """
3     To calculate OLS theta(s) given X, y in ndarrays.
4
5     Parameters:
6     -----
7         x: features, IV.
8         y: target variable, DV.
9         lambda: regularization parameter
10    Return:
11    -----

```

```

12     theta: coefficients
13     """
14
15     X = np.concatenate([np.ones((len(X), 1)), X], axis=1) # add x0 = 1 to matrix X
16     I = np.identity(len(X[0]))
17     theta = inv(X.T @ X + lambda * I) @ (X.T @ y)
18     return theta

```

```

1 len(X[0])

```

```

1 8

```

3. Comparison

Compare the two solutions in problem 1 and problem 2 and explain the reason why linear model with L2 regularization is robust. (using `climate_change_1.csv`)

```

1 X_train = df1_train.drop(df1_train.columns[8], axis=1).to_numpy()
2 y_train = df1_train.iloc[:, [8]].to_numpy()
3 theta_0 = closed_form_1(X_train, y_train)
4 theta_0

```

```

1 array([[ -1.24594260e+02],
2        [  6.42053134e-02],
3        [  6.45735927e-03],
4        [  1.24041896e-04],
5        [ -1.65280032e-02],
6        [ -6.63048889e-03],
7        [  3.80810324e-03],
8        [  9.31410835e-02],
9        [ -1.53761324e+00]])

```

```

1 theta_2 = closed_form_2(X_train, y_train, 0.5)
2 theta_2

```

```

1 array([[ -4.68953239e-03],
2        [  4.55768014e-02],
3        [  7.80443532e-03],
4        [  1.95701031e-04],
5        [ -1.64893727e-02],
6        [ -6.38359095e-03],
7        [  3.74766007e-03],
8        [  1.44919104e-03],
9        [ -3.65599605e-01]])

```

```

1 rsquare_test_theta_0 = score(y_test, predict(X_test, theta_0))
2 rsquare_test_theta_2 = score(y_test, predict(X_test, theta_2))
3 print("R2:", rsquare_test_theta_0, rsquare_test_theta_2)

```

```

1 R2: 0.22517701916248536 0.8022366128860412

```

Obviously, `theta_2`, which is the result of ridge regression, is much better due to the lower effect of redundant variables.

4. Change λ

You can change the regularization parameter λ to get different solutions for this problem. Suppose we set $\lambda = 10, 1, 0.1, 0.01, 0.001$, and please evaluate the model R^2 on the training set and the testing set.

```

1 from sklearn.metrics import mean_squared_error as mse
2
3 # Define constants
4 X_train = df1_train.drop(df1_train.columns[8], axis=1).to_numpy()
5 y_train = df1_train.iloc[:, [8]].to_numpy()
6 X_test = df1_test.drop(df1_test.columns[8], axis=1).to_numpy()
7 y_test = df1_test.iloc[:, [8]].to_numpy()

```



```

8 | lambds = [10.00, 1.000, 0.10, 0.01, 0.001]
9
10 | print("R scores comparison")
11 | # print("\t\t\t\t\t Training R2\t\t\t\t\t Testing R2\t\t\t\t\t Testing MSE")
12 | print("\t\t\t\t\t Training R2\t\t\t\t\t Testing R2")
13 | for lambd in lambds:
14 |     theta = closed_form_2(X_train, y_train, lambd)
15 |     rsquare_train = score(y_train, predict(X_train, theta))
16 |     rsquare_test = score(y_test, predict(X_test, theta))
17 |     # meanse = mse(y_test, predict(X_test, theta))
18 |     # print(lambd, "\t\t\t\t\t", rsquare_train.round(5), "\t\t\t\t\t", rsquare_test.round(5), "\t\t\t\t\t",
meanse.round(5))
19 |     print(lambd, "\t\t\t\t\t", rsquare_train.round(5), "\t\t\t\t\t", rsquare_test.round(5))

```

```

1 | R scores comparison
2 | λ\t\t\t\t\t Training R2\t\t\t\t\t Testing R2
3 | 10.0\t\t\t\t\t 0.67461\t\t\t\t\t 0.94087
4 | 1.0\t\t\t\t\t 0.67947\t\t\t\t\t 0.84675
5 | 0.1\t\t\t\t\t 0.69447\t\t\t\t\t 0.67329
6 | 0.01\t\t\t\t\t 0.71165\t\t\t\t\t 0.58528
7 | 0.001\t\t\t\t\t 0.71483\t\t\t\t\t 0.56252

```

Finally, please decide the best regularization parameter λ . (Note that: As a qualified data analyst, you must know how to choose model parameters, please learn about cross validation methods.)

```

1 | from sklearn.model_selection import KFold
2 | from sklearn.linear_model import Ridge
3 | from sklearn.model_selection import GridSearchCV
4
5 | def cross_validation(X, y):
6 |     """
7 |     Using k-fold to get optimal value of lambda based on R-squared.
8 |
9 |     Parameters:
10 |     -----
11 |         X: features, IV.
12 |         y: target variable, DV.
13 |
14 |     Return:
15 |     -----
16 |         alpha: learning rate
17 |     """
18 |     kfold = KFold(n_splits=10).split(X, y)
19 |     model = Ridge(normalize=True) # Normalization returns better result
20 |     lambdas = [10, 1, 0.1, 0.01, 0.001]
21 |     grid_param = {"alpha": lambdas}
22 |     grid = GridSearchCV(estimator=model,
23 |                         param_grid=grid_param,
24 |                         cv=kfold,
25 |                         scoring="r2")
26 |     grid.fit(X, y)
27 |     alpha = grid.best_params_['alpha']
28 |     return alpha
29
30 | print('Optimal lambda should be ', cross_validation(X_train, y_train))

```

```

1 | Optimal lambda should be 0.1

```

```

1 | C:\Users\oyrx\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:814: DeprecationWarning: The
default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This
will change numeric results when test-set sizes are unequal.
2 | DeprecationWarning)

```

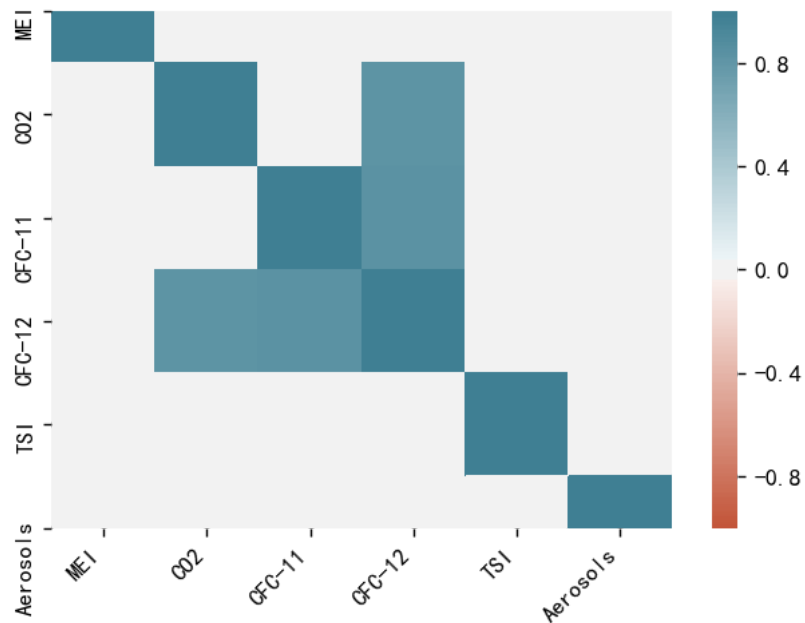
Problem 3 — Feature Selection

1. Lesser variables

From Problem 1, you can know which variables are significant, therefore you can use less variables to train model. For example, remove highly correlated and redundant features. You can propose a workflow to select feature.

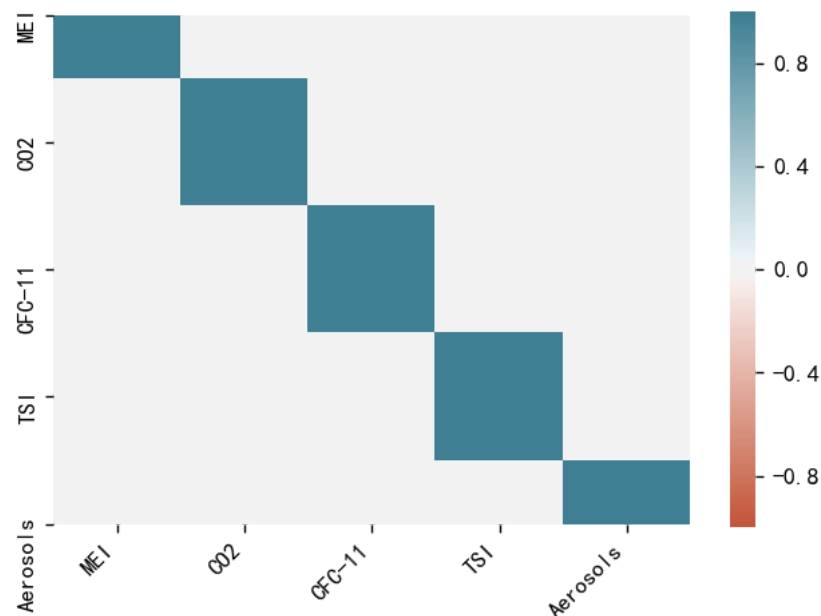
As mentioned in the first section and known significant variables(MEI, CO2, CFC-11, CFC-12, TSI, Aerosols), a new correlation matrix can be introduced:

```
1 corr = df1[['MEI', 'CO2', 'CFC-11', 'CFC-12', 'TSI', 'Aerosols']].corr()
2 high_corr = corr[np.abs(corr) > 0.5].fillna(0)
3 corr[np.abs(corr) > 0.6].fillna('')
4 plt.figure(dpi=128)
5 ax = sns.heatmap(
6     high_corr,
7     vmin=-1, vmax=1, center=0,
8     cmap=sns.diverging_palette(20, 220, n=200),
9     square=True
10 )
11 ax.set_xticklabels(
12     ax.get_xticklabels(),
13     rotation=45,
14     horizontalalignment='right'
15 );
```



Thus, CFC-12 should also be removed($r > 0.6$) then we have:

```
1 corr = df1[['MEI', 'CO2', 'CFC-11', 'TSI', 'Aerosols']].corr()
2 high_corr = corr[np.abs(corr) > 0.5].fillna(0)
3 corr[np.abs(corr) > 0.6].fillna('')
4 plt.figure(dpi=128)
5 ax = sns.heatmap(
6     high_corr,
7     vmin=-1, vmax=1, center=0,
8     cmap=sns.diverging_palette(20, 220, n=200),
9     square=True
10 )
11 ax.set_xticklabels(
12     ax.get_xticklabels(),
13     rotation=45,
14     horizontalalignment='right'
15 );
```



Now no redundant variables left.

2. A better model

Train a better model than the model in Problem 2.

```
1 x_lesser = df1_train[['MEI', 'CO2', 'CFC-11', 'TSI', 'Aerosols']].to_numpy() # x: the features
2 y_lesser = df1_train.iloc[:, [8]].to_numpy() # y: the results, lower case to
  emphasize the difference
3 x_test = df1_test[['MEI', 'CO2', 'CFC-11', 'TSI', 'Aerosols']].to_numpy()
4 y_test = df1_test.iloc[:, [8]].to_numpy()
5
6 #theta_lesser = closed_form_1(x_lesser, y_lesser)
7 theta_lesser = closed_form_2(x_lesser, y_train, cross_validation(x_lesser, y_lesser))
8 theta_lesser = np.array(theta_lesser)
9 formula = [str(theta_lesser.round(5).tolist()[i][0]) + ' * x' + str(i) + ' + ' for i in range(0,
  len(theta_lesser.round(5).tolist()))]
10 print('Thus our better model is: \nny = '+' '.join(formula).replace(' * x0', '')[:-3])
```

```
1 Thus our better model is:
2
3 y = -0.02465 + 0.04909 * x1 + 0.0118 * x2 + 2e-05 * x3 + -0.00293 * x4 + -0.88807 * x5
```

```
1 C:\Users\oyrx\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:814: DeprecationWarning: The
  default of the 'iid' parameter will change from True to False in version 0.22 and will be removed in 0.24. This
  will change numeric results when test-set sizes are unequal.
2 DeprecationWarning)
```

Significance:

```
1 l = sm.OLS(y_lesser, x_lesser).fit()
2 pvalues = l.summary2().tables[1]['P>|t|']
3 pvalues<0.05
```

```
1 x1    True
2 x2    True
3 x3    False
4 x4    True
5 x5    True
6 Name: P>|t|, dtype: bool
```

Then remove x3 based on the new result:

```

1 x_lesser = df1_train[['MEI', 'CO2', 'TSI', 'Aerosols' ]].to_numpy() # x: the features
2 y_lesser = df1_train.iloc[:, [8]].to_numpy() # y: the results, lower case to
  emphasize the difference
3 x_test = df1_test[['MEI', 'CO2', 'TSI', 'Aerosols' ]].to_numpy()
4 y_test = df1_test.iloc[:, [8]].to_numpy()
5
6 theta_lesser = closed_form_1(x_lesser, y_train)
7 theta_lesser = np.array(theta_lesser)
8 formula = [str(theta_lesser.round(5).tolist()[i][0]) + ' * x' + str(i) + ' + ' for i in range(0,
  len(theta_lesser.round(5).tolist()))]
9 print('Thus our better model is: \n\nty = '+ ' '.join(formula).replace(' * x0', '')[:-3])

```

```

1 Thus our better model is:
2
3 y = -118.60162 + 0.06204 * x1 + 0.01069 * x2 + 0.08418 * x3 + -1.58444 * x4

```

```

1 l = sm.OLS(y_lesser, x_lesser).fit()
2 pvalues = l.summary2().tables[1]['P>|t|']
3 pvalues<0.05

```

```

1 x1    True
2 x2    True
3 x3    True
4 x4    True
5 Name: P>|t|, dtype: bool

```

R2:

```

1 rsquare_train = score(y_lesser, predict(x_lesser, theta_lesser))
2 rsquare_test = score(y_test, predict(x_test, theta_lesser))
3 print(('R2\nTraining: {}\nTesting: {}'.format(rsquare_train, rsquare_test))

```

```

1 R2
2 Training: 0.7336403428986277
3 Testing: 0.6328867941215359

```

Problem 4 — Gradient Descent

Gradient descent algorithm is an iterative process that takes us to the minimum of a function. Please write down the iterative expression for updating the solution of linear model and implement it using Python or Matlab in gradientDescent function.

Define functions

```

1 def cost_function(X: np.ndarray, y: np.ndarray, theta: np.ndarray) -> float:
2     """
3     To calculate cost given X, y, and theta in ndarrays.
4
5     Parameters:
6     -----
7         x: features, IV.
8         y: target variable, DV.
9         theta: coefficients
10    Return:
11    -----
12        cost: calculated cost
13    """
14    cost = np.sum(np.power((X * theta.T) - y, 2)) / (2 * len(X))
15    return cost
16
17
18 def gradientDescent(X: np.ndarray, y: np.ndarray, theta: np.ndarray,
19                    alpha: float, iters: float) -> np.matrix and np.ndarray:
20     """
21     To find theta given x, y, theta in ndarrays and alpha, iters in float.

```

```

22
23     Parameters:
24     -----
25         x: features, IV.
26         y: target variable, DV.
27         theta: coefficients
28         alpha: learning rate
29         iters: an assigned number of iterations
30     Return:
31     -----
32         theta: np.matrix, final theta
33         cost: np.ndarray, calculated cost
34     """
35
36     temp = np.mat(np.zeros(theta.shape))
37     cost = np.zeros(iters)
38     thetaNums = int(theta.shape[1])
39
40     for i in range(iters):
41         error = (X * theta.T - y)
42         for j in range(thetaNums):
43             derivativeInner = np.multiply(error, X[:, j])
44             temp[0, j] = theta[0, j] - (alpha * np.sum(derivativeInner) /
45                                                         len(X))
46
47         theta = temp
48         cost[i] = cost_function(X, y, theta)
49     return theta, cost

```

Define parameters

```

1  import matplotlib.pyplot as plt
2
3
4  theta = np.mat([0,0,0,0,0,0,0,0,0])
5  iters = 5000 # The number of iterations
6  alpha = 0.01 # Learning rate
7
8  x = df1.get(["MEI", "CO2", "CH4", "N2O", "CFC-11", "CFC-12", "TSI", "Aerosols"])
9  x = np.column_stack((np.ones(len(x)), x))
10 x_train = x[:284]
11 x_test = x[284:]
12 x_train = np.mat(x_train)
13
14 y = df1.get("Temp")
15 y_train = y[:284]
16 y_test = y[284:]
17
18 y_train = np.mat(y_train).T
19
20 for i in range(1,9):
21     x_train[:,i] = (x_train[:,i] - min(x_train[:,i])) / (max(x_train[:,i]) - min(x_train[:,i]))
22
23 theta_n = (x_train.T*x_train).I*x_train.T*y_train
24 print("theta =", theta_n)

```

```

1  theta = [[-0.07698894]
2           [ 0.29450977]
3           [ 0.28935427]
4           [ 0.02211171]
5           [-0.27724073]
6           [-0.53156629]
7           [ 0.7376296 ]
8           [ 0.17604596]
9           [-0.22725924]]

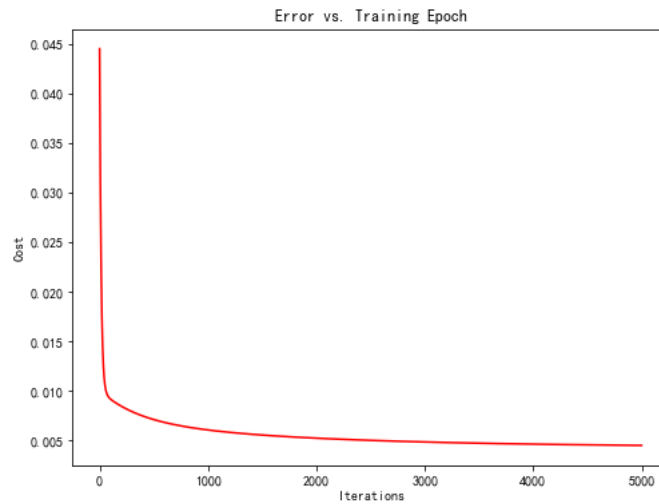
```

Find theta

```
1 finalTheta, cost = gradientDescent(X_train, y_train, theta, alpha, iters)
```

Check the result

```
1 fig, bx = plt.subplots(figsize=(8,6))
2 bx.plot(np.arange(iters), cost, 'r')
3 bx.set_xlabel('Iterations')
4 bx.set_ylabel('Cost')
5 bx.set_title('Error vs. Training Epoch')
6 plt.show()
```



The result indicates that algorithm has converged and the final theta is

```
1 print("Final theta ", finalTheta, 'with cost', cost)
```

```
1 Final theta [[-0.07049773  0.19881777  0.19132161  0.06760065  0.18431036 -0.08700985
2              0.08339899  0.1001724  -0.16872269]] with cost [0.0444738  0.04207593 0.0398436  ... 0.0044812  0.0044811
0.004481  ]
```

```
1
```