FinalProject_WeatherData

April 17, 2022

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

```
[2]: #import weather data
     weather = pd.read_csv('canada_weather.csv')
     weather.head()
```

```
[2]:           Community Weather station  \
     0      Alberton, PE             NaN
     1    Baker Lake, NU             YBK
     2   Baie-Comeau, QC             YBC
     3       Calgary, AB             YYC
     4  Charlottetown, PE             YYG

                                                 Location          Elevation  \
     0  46°51 00 N 064°01 00 W / 46.85000°N 64.01667°W…         3m (9.8ft)
     1  64°17 56 N 096°04 40 W / 64.29889°N 96.07778°W…      18.6m (61ft)
     2  49°08 00 N 068°12 00 W / 49.13333°N 68.20000°W…         22m (72ft)
     3  51°06 50 N 114°01 13 W / 51.11389°N 114.02028°…  1,084m (3,556ft)
     4  46°17 19 N 063°07 43 W / 46.28861°N 63.12861°W…       49m (161ft)

       January(Avg. high °C (°F)) January(Avg. low °C (°F))  \
     0              -3.9 (25.0)             -12.5 (9.5)
     1            -27.7 (-17.9)           -34.8 (-30.6)
     2              -8.7 (16.3)            -19.9 (-3.8)
     3              -0.9 (30.4)             -13.2 (8.2)
     4              -3.4 (25.9)            -12.1 (10.2)

       July(Avg. high °C (°F)) July(Avg. low °C (°F)) Annual(Avg. high °C (°F))  \
     0            23.2 (73.8)          14.1 (57.4)                9.6 (49.3)
     1            17.0 (62.6)           6.1 (43.0)               -7.3 (18.9)
     2            20.9 (69.6)          10.3 (50.5)                6.6 (43.9)
     3            23.2 (73.8)           9.8 (49.6)               10.8 (51.4)
     4            23.3 (73.9)          14.1 (57.4)                9.9 (49.8)
```

```
  Annual(Avg. low °C (°F))
0               1.3 (34.3)
1              -15.2 (4.6)
2              -3.3 (26.1)
3              -1.9 (28.6)
4               1.3 (34.3)
```

[3]:

```
#split location and get latitude and longitude
# and then separate out latitude
weather['Latitude']=weather['Location'].apply(lambda x : x.split('/')[2]).
 ↪apply(lambda x : x.split(';')[0])
print(weather['Latitude'].head())
```

```
0     46.85000
1     64.29889
2     49.13333
3     51.11389
4     46.28861
Name: Latitude, dtype: object
```

[4]:

```
#split location and get latitude and longitude
# and then separate out latitude
weather['Longitude']=weather['Location'].apply(lambda x : x.split('/')[2]).
 ↪apply(lambda x : x.split(';')[1])
print(weather['Longitude'].head())
```

```
0               -64.01667 (Alberton)
1     -96.07778 (Baker Lake Airport)
2            -68.20000 (Baie-Comeau)
3               -114.02028 (Calgary)
4          -63.12861 (Charlottetown)
Name: Longitude, dtype: object
```

[5]:

```
#separate just the longitude
weather['Longitude']=weather['Longitude'].apply(lambda x : x.split(' (')[0].
 ↪strip())
```

[6]:

```
#converting to integer from string
weather['Longitude']=[float(long) for long in weather['Longitude']]
```

[7]:

```
#taking just the °C value for annual avg low
weather['Annual(Avg. low °C (°F))']=weather['Annual(Avg. low °C (°F))'].
 ↪apply(lambda x : x.split(' (')[0])
```

[8]:

```
#taking just the °C value for annual avg low
weather['Annual(Avg. high °C (°F))']=weather['Annual(Avg. high °C (°F))'].
 ↪apply(lambda x : x.split(' (')[0])
```

[9]:

```
#taking elevation in meters
weather['Elevation']=weather['Elevation'].apply(lambda x : x.split(' (')[0])
```

[10]:

```
#removing meter unit from elevation
weather['Elevation']=weather['Elevation'].str.replace("m","")
```

[11]:

```
#converting to integer from string
import re
weather['Elevation']=[int(float(re.sub(",", "", e))) for e in␣
 ↪weather['Elevation']]
```

[12]:

```
# renaming the column by removing °F
weather.rename(columns = {'Annual(Avg. low °C (°F))':'Annual(Avg. low °C)'},␣
 ↪inplace=True)
weather.rename(columns = {'Annual(Avg. high °C (°F))':'Annual(Avg. high °C)'},␣
 ↪inplace=True)
```

[13]:

```
weather['Annual(Avg. high °C)']=[float(re.sub("-", "-", high)) for high in␣
 ↪weather['Annual(Avg. high °C)']]
```

[14]:

```
weather['Annual(Avg. low °C)']=[float(re.sub("-", "-", low)) for low in␣
 ↪weather['Annual(Avg. low °C)']]
```

[15]:

```
# adding unit to the column Elevation
weather.rename(columns = {'Elevation':'Elevation(m)'}, inplace=True)
```

[16]:

```
#dropping unwanted columns
weather.drop(['Community','Weather station','Location','January(Avg. high °C␣
 ↪(°F))','January(Avg. low °C (°F))','July(Avg. high °C (°F))','July(Avg. low␣
 ↪°C (°F))'], axis=1, inplace=True)
```

[17]:

```
#rearranging columns
weather = weather[['Latitude','Longitude','Elevation(m)','Annual(Avg. high␣
 ↪°C)','Annual(Avg. low °C)']]
```

[18]:

```
weather.head()
```

[18]:

```
    Latitude  Longitude  Elevation(m)  Annual(Avg. high °C)  \
0   46.85000  -64.01667             3                   9.6
1   64.29889  -96.07778            18                  -7.3
2   49.13333  -68.20000            22                   6.6
3   51.11389 -114.02028          1084                  10.8
4   46.28861  -63.12861            49                   9.9

   Annual(Avg. low °C)
0                  1.3
1                -15.2
```

```
2                   -3.3
3                   -1.9
4                    1.3
```

[19]:

```
weather_subset=weather[['Elevation(m)','Annual(Avg. high °C)','Annual(Avg. low␣
  ↪°C)']]
print(weather_subset.head())
```

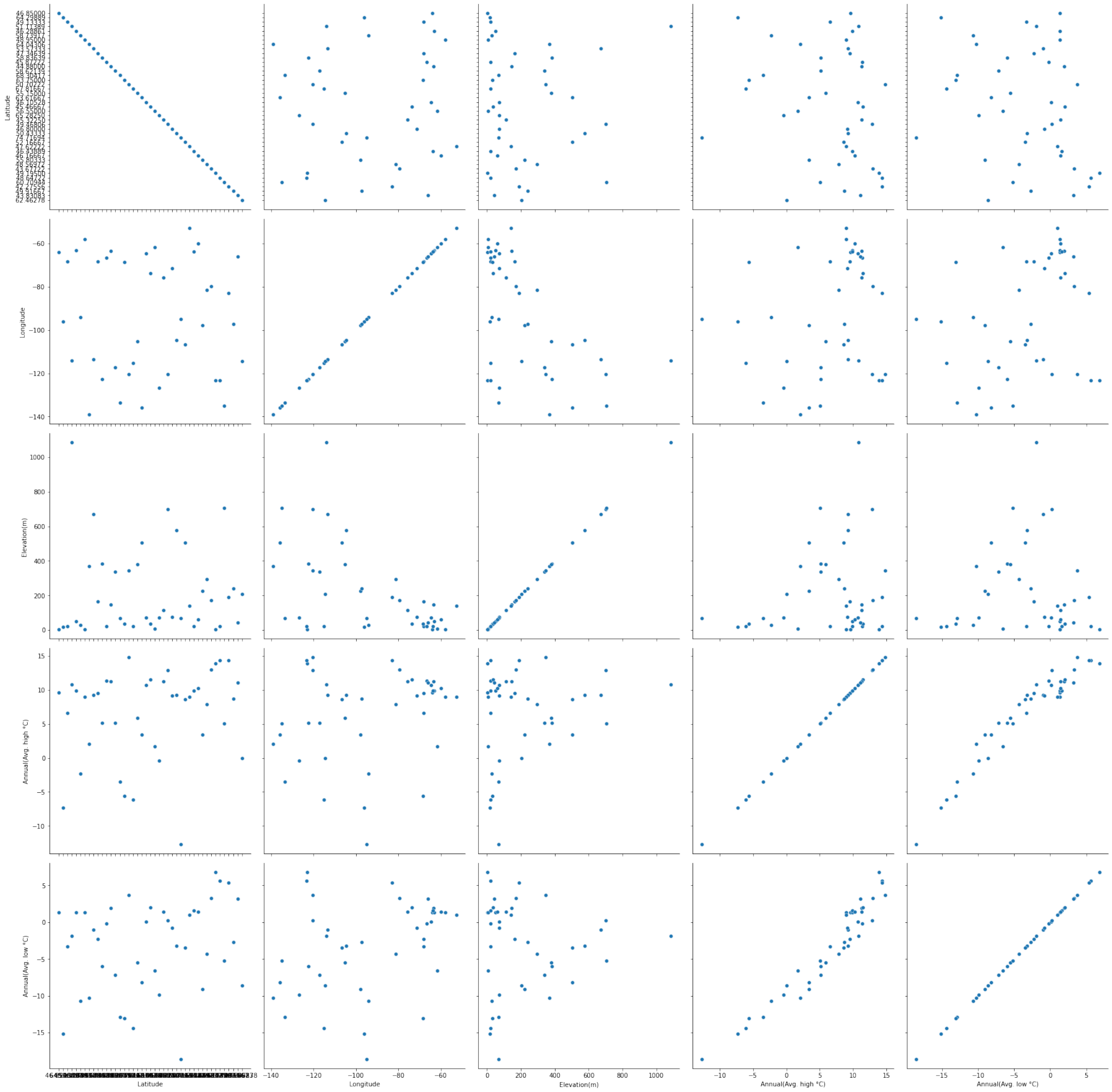
```
   Elevation(m)  Annual(Avg. high °C)  Annual(Avg. low °C)
0             3                   9.6                  1.3
1            18                  -7.3                -15.2
2            22                   6.6                 -3.3
3          1084                  10.8                 -1.9
4            49                   9.9                  1.3
```

[20]:

```
x = sns.PairGrid(weather, vars=weather,  height = 5)
x.map(sns.scatterplot)
```

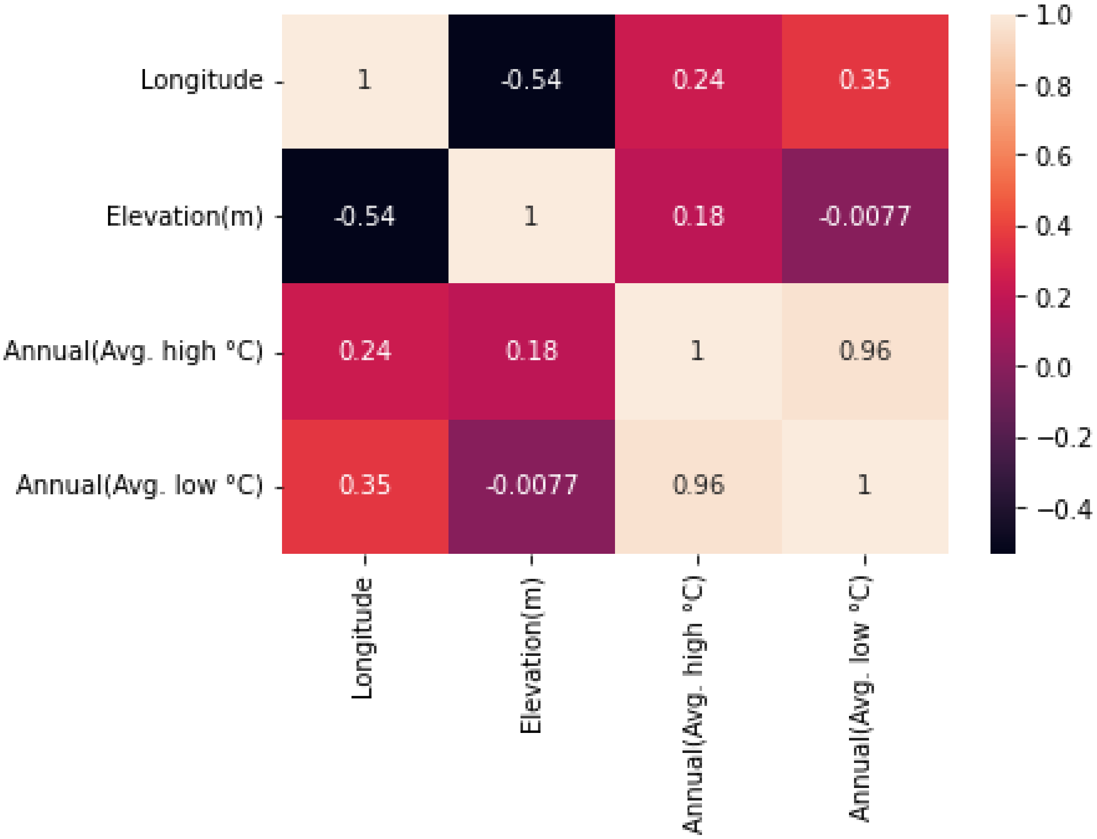
[20]:

```
<seaborn.axisgrid.PairGrid at 0x1e0584fad70>
```



```
[21]: sns.heatmap(weather.corr(),annot=True)
```

```
[21]: <AxesSubplot:>
```



[22]:

```
X=weather[['Latitude','Longitude','Elevation(m)']].values
y=weather['Annual(Avg. high °C)'].values
print(X)
print(y)
```

```
[[' 46.85000' -64.01667 3]
 [' 64.29889' -96.07778 18]
 [' 49.13333' -68.2 22]
 [' 51.11389' -114.02028 1084]
 [' 46.28861' -63.12861 49]
 [' 58.73917' -94.06639 29]
 [' 48.95000' -57.95 5]
 [' 64.04306' -139.12778 370]
 [' 53.57333' -113.51833 671]
 [' 47.34639' -68.18778 163]
 [' 58.83639' -122.59722 382]
 [' 45.87222' -66.52778 21]
 [' 44.88000' -63.5 145]
 [' 58.62139' -117.16472 338]
 [' 68.30417' -133.48278 68]
 [' 63.75000' -68.55 34]
 [' 50.70222' -120.44194 345]
 [' 67.81667' -115.14389 23]
 [' 55.15000' -105.26667 379]
 [' 63.61667' -135.86667 504]
 [' 46.10528' -64.68389 71]
 [' 45.46667' -73.75 36]
 [' 56.55000' -61.68333 6]
 [' 65.28250' -126.80028 73]
 [' 45.32250' -75.66917 114]
 [' 49.46806' -120.51139 700]
 [' 46.80000' -71.38333 74]
 [' 50.43333' -104.66667 578]
 [' 74.71694' -94.96944 68]
 [' 52.16667' -106.71667 504]
 [' 47.62222' -52.74278 141]
 [' 46.43889' -63.83167 20]
 [' 46.16667' -60.04806 62]
 [' 55.80333' -97.8625 224]
 [' 48.56972' -81.37667 295]
 [' 43.67722' -79.63056 173]
 [' 49.19500' -123.18194 4]
 [' 48.64722' -123.42583 20]
 [' 60.70944' -135.06889 706]
 [' 42.27556' -82.95556 190]
 [' 49.91667' -97.23333 239]
 [' 43.83083' -66.08861 43]
 [' 62.46278' -114.44028 206]]
[  9.6  -7.3   6.6  10.8   9.9  -2.3   9.    2.1   9.3   9.5   5.2  11.4
  11.3   5.2  -3.5  -5.6  14.8  -6.1   5.9   3.4  10.7  11.5   1.7  -0.4
  11.3  12.9   9.2   9.3 -12.7   8.6   9.    9.9  10.3   3.4   7.9  13.
  13.9  14.4   5.1  14.4   8.7  11.1   0. ]
```

[23]:

```python
from sklearn.model_selection import train_test_split
print(X.shape,y.shape)
```

```
(43, 3) (43,)
```

[24]:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(32, 3) (11, 3) (32,) (11,)
```

[25]:

```python
#Polynomial Regression for Annual high temperature
X = weather[['Latitude','Longitude','Elevation(m)']]
y_high = weather['Annual(Avg. high °C)']

X_high_train, X_high_test, y_high_train, y_high_test = train_test_split(X,␣
 ↪y_high, test_size=0.2, random_state=101)
```

```
print(X_high_train.shape,X_high_test.shape,X.shape)
```

```
(34, 3) (9, 3) (43, 3)
```

[26]:

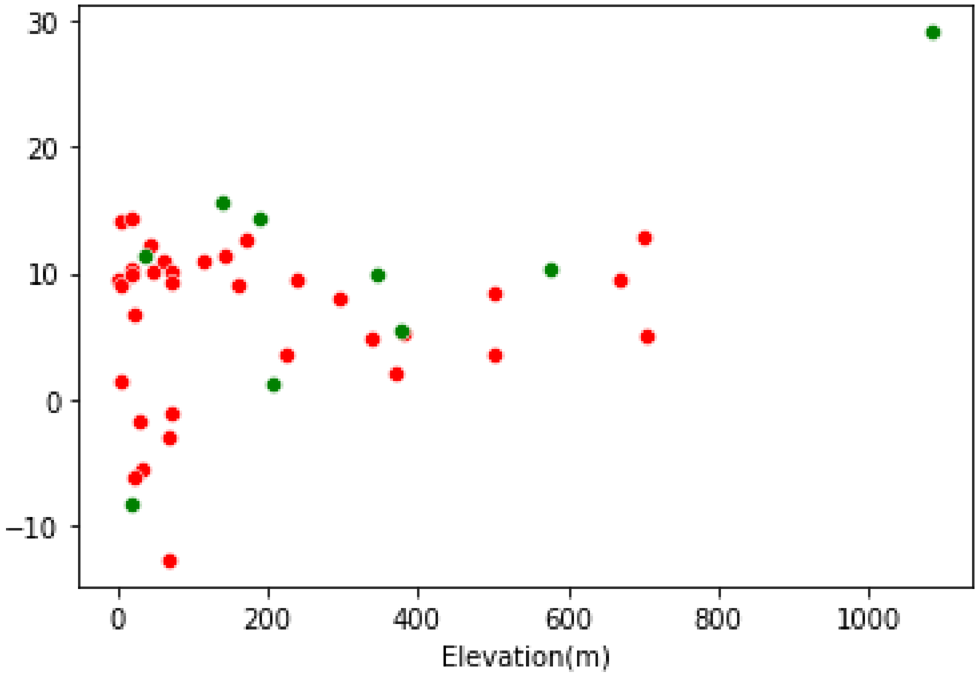
```
## builiding model
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
poly_model = make_pipeline(PolynomialFeatures(degree=3,include_bias=True),
                           LinearRegression())
# evaluation
poly_model.fit(X_high_train, y_high_train)
yfit_high = poly_model.predict(X_high_test)
yfit_high_training = poly_model.predict(X_high_train)

sns.scatterplot(X_high_train['Elevation(m)'],yfit_high_training,color='r') #␣
 ↪ground_truth
sns.scatterplot(X_high_test['Elevation(m)'],yfit_high,color='g') # prediction
```

```
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[26]:

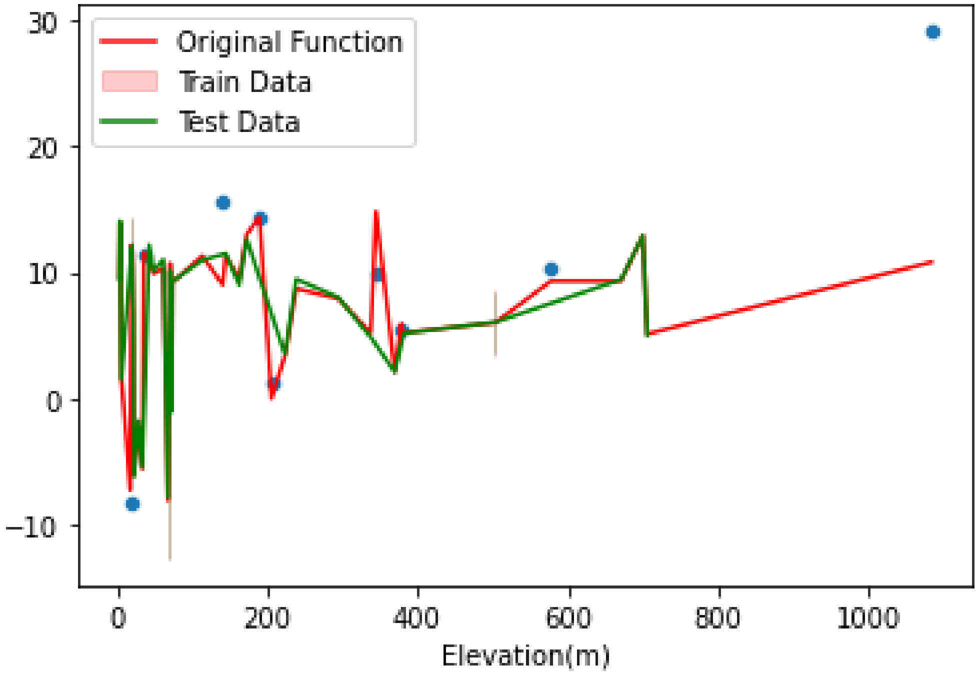
```
<AxesSubplot:xlabel='Elevation(m)'>
```



[27]:
```
sns.lineplot(X.iloc[:,2],y,color='r')
sns.lineplot(X_high_train.iloc[:,2],yfit_high_training,color='g')
sns.scatterplot(X_high_test.iloc[:,2],yfit_high)

plt.legend(labels=['Original Function','Train Data','Test Data'])
```

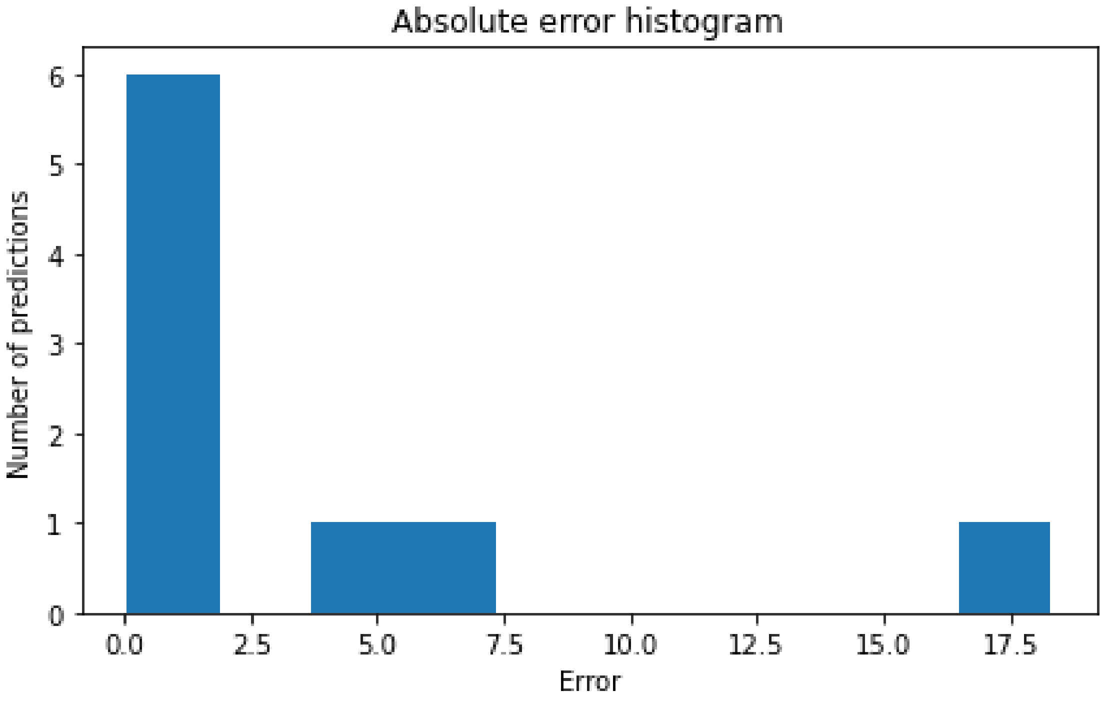
```
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
[27]: <matplotlib.legend.Legend at 0x1e05ccd64d0>
```



```
[28]: from sklearn import metrics
      print('MAE:', metrics.mean_absolute_error(y_high_test, yfit_high))
      print('RMSE Test:', np.sqrt(metrics.mean_squared_error(y_high_test, yfit_high)))
      print('RMSE Training:', np.sqrt(metrics.mean_squared_error(y_high_train,␣
       ↪yfit_high_training)))
```

```
MAE: 3.740632398582406
RMSE Test: 6.7079685692302675
RMSE Training: 0.4096556319574178
```

```
[29]: absError=abs((yfit_high-y_high_test))
      plt.figure(figsize=(7,4))
      plt.xlabel("Error")
      plt.ylabel("Number of predictions")
      plt.title("Absolute error histogram")
      plt.hist(absError)
      plt.show()
```



[30]:

```
#Polynomial Regression for Annual low temperature
X = weather[['Latitude','Longitude','Elevation(m)']]
y_low = weather['Annual(Avg. low °C)']

X_low_train, X_low_test, y_low_train, y_low_test = train_test_split(X, y_low,␣
 ↪test_size=0.2, random_state=101)
print(X_low_train.shape,X_low_test.shape,X.shape)
```

```
(34, 3) (9, 3) (43, 3)
```

[31]:

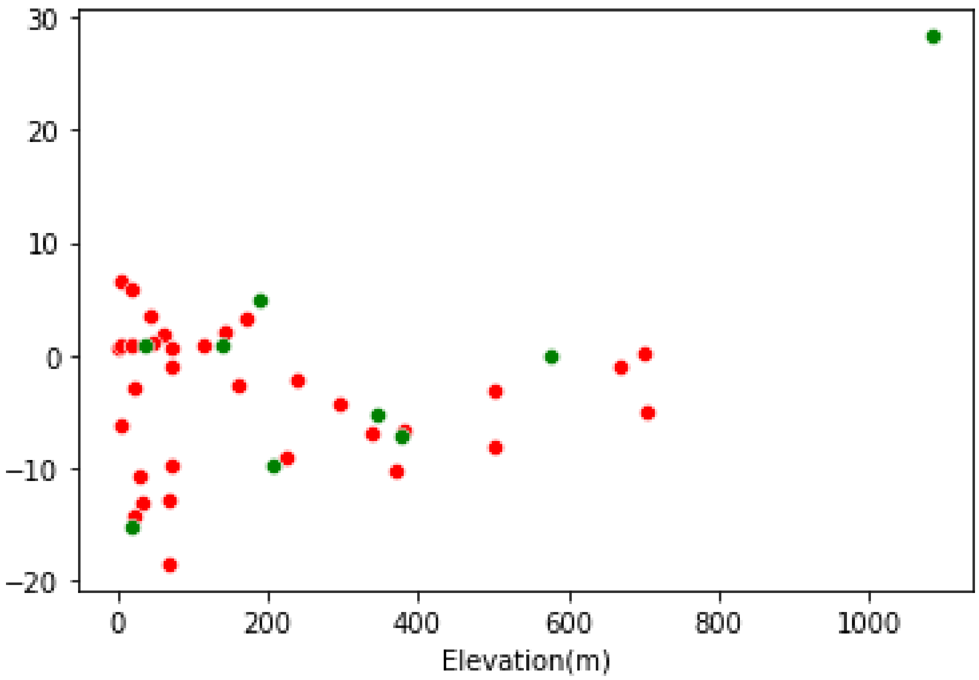
```
## builiding model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
poly_model = make_pipeline(PolynomialFeatures(degree=3,include_bias=True),
                           LinearRegression())
# evaluation
poly_model.fit(X_low_train, y_low_train)
yfit_low = poly_model.predict(X_low_test)
yfit_low_training = poly_model.predict(X_low_train)

sns.scatterplot(X_low_train['Elevation(m)'],yfit_low_training,color='r') #␣
 ↪ground_truth
sns.scatterplot(X_low_test['Elevation(m)'],yfit_low,color='g') # prediction
```

```
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
```

```
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

```
[31]: <AxesSubplot:xlabel='Elevation(m)'>
```



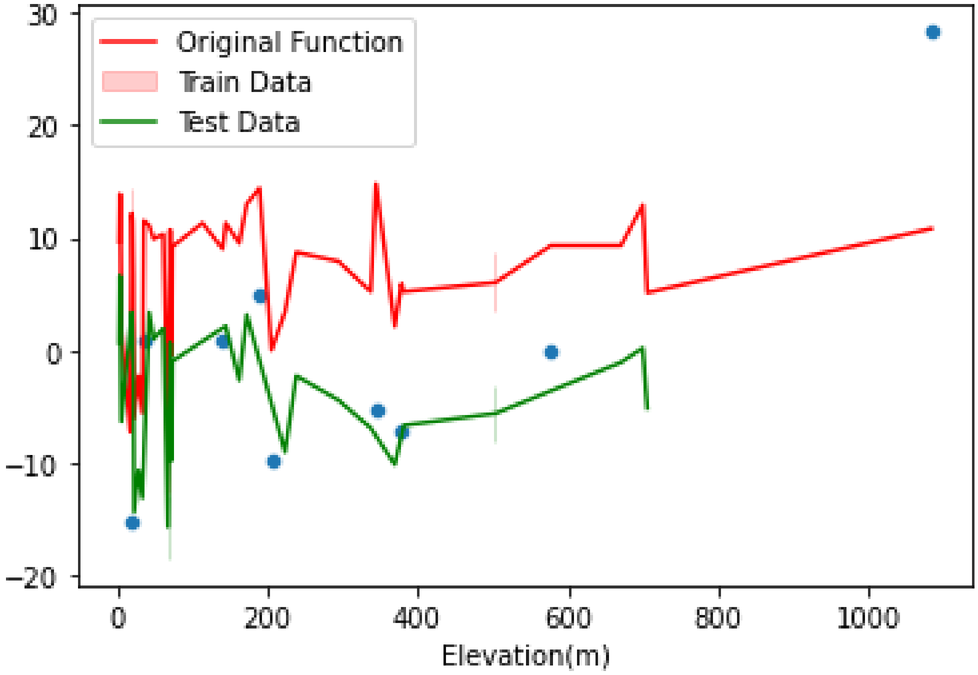
```
[32]: sns.lineplot(X.iloc[:,2],y,color='r')
      sns.lineplot(X_low_train.iloc[:,2],yfit_low_training,color='g')
      sns.scatterplot(X_low_test.iloc[:,2],yfit_low)

      plt.legend(labels=['Original Function','Train Data','Test Data'])
```

```
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
```

```
  warnings.warn(
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
C:\Users\priya\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

```
[32]: <matplotlib.legend.Legend at 0x1e0603df100>
```

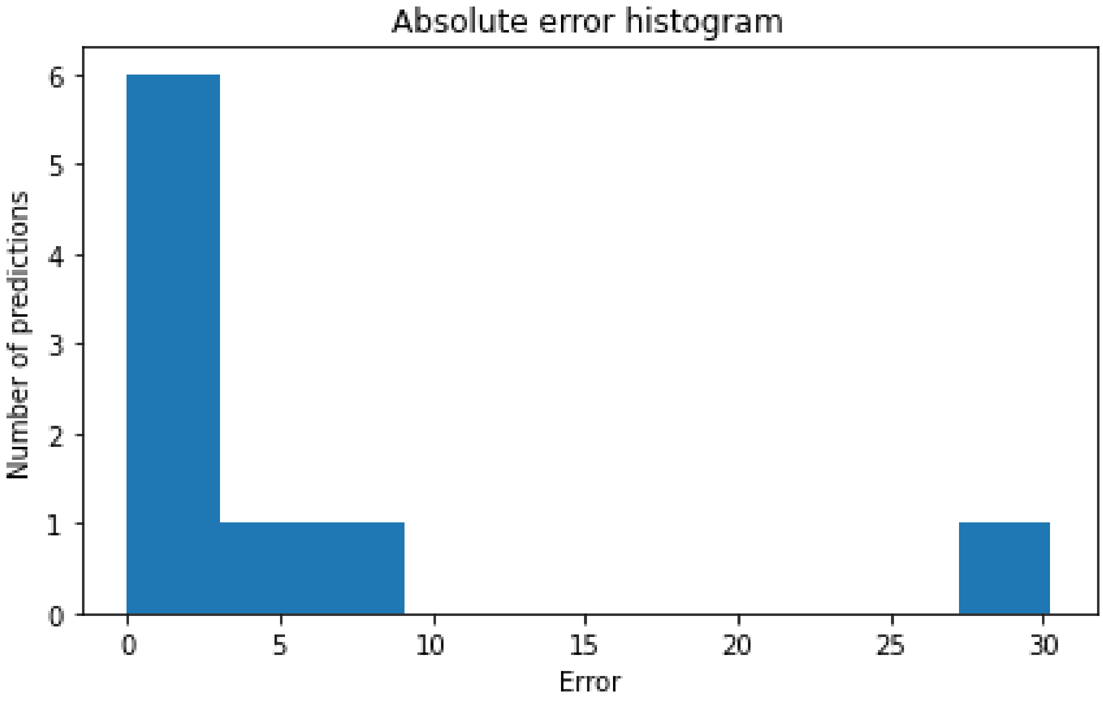


```
[33]: print('MAE:', metrics.mean_absolute_error(y_low_test, yfit_low))
      print('RMSE Test:', np.sqrt(metrics.mean_squared_error(y_low_test, yfit_low)))
      print('RMSE Training:', np.sqrt(metrics.mean_squared_error(y_low_train,␣
        ↪yfit_low_training)))
```

```
MAE: 5.244826449721522
RMSE Test: 10.600822745470317
RMSE Training: 0.38415976396258084
```

[34]:

```
absError1=abs((yfit_low-y_low_test))
plt.figure(figsize=(7,4))
plt.xlabel("Error")
plt.ylabel("Number of predictions")
plt.title("Absolute error histogram")
plt.hist(absError1)
plt.show()
```



[ ]:

```

```