In [1]:

**import** matplotlib.pyplot **as** plt  
**%matplotlib** inline  
**import** math  
**import** numpy **as** np  
**import** pandas **as** pd  
**import** scipy.stats **as** stats  
**import** statsmodels.api **as** sm  
**import** statsmodels.stats.api **as** sms  
**import** statsmodels.formula.api **as** smf  
**import** statsmodels.stats.multicomp **as** smm  
**import** statsmodels.stats.libqsturng **as** slb  
**import** statsmodels  
**import** warnings  
plt**.**style**.**use("default")  
plt**.**rcParams**.**update({"figure.dpi": 70, "axes.titlesize": 16, "axes.labelsize": 13, "font.size": 11})

In [2]:

**class** myANOVA:  
 **def** \_\_init\_\_(self, data, method**=**"one-way"): *# input needs to be melted*  
 **if** method **not** **in** ["one-way", "rand-block", "two-factor"]:  
 **raise** NotImplementedError("Error: Only supports 'one-way', 'rand-block', or 'two-factor' ANOVA")  
 **if** "Value" **not** **in** data**.**columns:  
 **raise** RuntimeError("Error: 'Value' is not a column in data.")  
   
 self**.**data **=** data**.**copy()  
 self**.**method **=** method  
 self**.**anova\_table **=** pd**.**DataFrame()  
   
   
 **def** data\_summary(self, by):  
 print("="**\***70)  
 print("Descriptive Statistics for All Treatments:")  
 display(self**.**data**.**groupby(by)["Value"]**.**describe()[["count", "mean", "std"]])  
   
   
 **def** required\_condition\_summary(self, by):  
 cols **=** list(sorted(set(self**.**data[by])))  
   
 self**.**data**.**hist(by**=**by, column**=**"Value", bins**=**5) *# Histogram*  
 print("="**\***70)  
 print("Histograms for All Treatments:")  
 plt**.**show()  
   
 **for** i, col **in** enumerate(cols): *# Q-Q Plot*  
 **if** i**%2** == 0: fig = plt.figure(figsize=(9.5, 4))  
 ax **=** plt**.**subplot(1, 2, i**%2** + 1)  
 sm**.**qqplot(self**.**data[self**.**data[by] **==** col]["Value"]**.**dropna(), stats**.**norm, fit**=True**, line**=**'45', ax**=**ax)  
 ax**.**set\_box\_aspect(1)  
 plt**.**title(f"Q-Q Plot for {col}")  
 **if** i**%2** == 1 or i == len(cols) - 1: plt.show()  
   
 print("Shapiro Test:") *# Shapiro Test*  
 sh\_df **=** pd**.**DataFrame(columns**=**cols, index**=**["shapiro stat", "p value"])  
 **for** col **in** cols:  
 shapiro\_stat, sh\_p\_val **=** stats**.**shapiro(self**.**data[self**.**data[by] **==** col]["Value"]**.**dropna())  
 sh\_df[col]["shapiro stat"] **=** shapiro\_stat  
 sh\_df[col]["p value"] **=** sh\_p\_val  
 **if** sh\_p\_val **<** .05: warnings**.**warn(f"Warning: Shapiro Test suggests that the distribution of {col} is not normal. (alpha=.05)")  
 display(sh\_df)  
   
 print("Bartlett Test:") *# Bartlett Test*  
 ba\_df **=** pd**.**DataFrame(columns**=**["All Treatments"], index**=**["bartlett stat", "p value"])  
 bartlett\_stat, bartlett\_p\_val **=** stats**.**bartlett(**\***[self**.**data[self**.**data[by] **==** col]["Value"]**.**dropna() **for** col **in** cols])  
 ba\_df["All Treatments"]["bartlett stat"] **=** bartlett\_stat  
 ba\_df["All Treatments"]["p value"] **=** bartlett\_p\_val  
 **if** bartlett\_p\_val **<** .05: warnings**.**warn(f"Warning: Bartlett Test suggests that the variances of treatments are unequal. (alpha=.05)")  
 display(ba\_df)  
   
   
 **def** anova\_summary(self, by, hide\_result**=False**):  
 **if** **not** hide\_result: print("="**\***70)  
 **if** self**.**method **==** "one-way": *# One-Way ANOVA*  
 **if** **not** hide\_result:  
 print("One-Way ANOVA Table:")  
 data\_ols **=** smf**.**ols(f"Value ~ C({by})", data**=**self**.**data)**.**fit()  
 **elif** self**.**method **==** "rand-block": *# randomized block ANOVA*  
 **if** **not** hide\_result:  
 print("Randomized Block ANOVA Table:")  
 data\_ols **=** smf**.**ols(f"Value ~ C({by[0]}) + C({by[1]})", data**=**self**.**data)**.**fit()  
 **else**: *# Two-Factor ANOVA*  
 **if** **not** hide\_result:  
 print("Two-Factor ANOVA Table:")  
 data\_ols **=** smf**.**ols(f"Value ~ C({by[0]}) + C({by[1]}) + C({by[0]}):C({by[1]})", data**=**self**.**data)**.**fit()  
 anova\_table **=** sms**.**anova\_lm(data\_ols, typ**=**2)  
 anova\_table**.**insert(2, "mean\_sq", anova\_table["sum\_sq"] **/** anova\_table["df"]) *# compute MST and MSE*  
 anova\_table**.**loc["Total"] **=** anova\_table**.**sum(axis**=**0, skipna**=False**) *# compute SS and df*  
 anova\_table["mean\_sq"]["Total"] **=** np**.**nan  
 **if** **not** hide\_result: display(anova\_table)  
 self**.**anova\_table **=** anova\_table  
   
   
 **def** multiple\_comp\_summary(self, by, alpha, method**=**"Tukey"):  
 **if** self**.**method **!=** "one-way":  
 **raise** RuntimeError("Error: Multiple comparisons available only with one-way ANOVA.")  
 **if** method **not** **in** ["Fisher", "Tukey"]:  
 **raise** NotImplementedError("Error: method must be 'Tukey' or 'Fisher'.")  
 **if** self**.**anova\_table**.**empty:  
 **raise** RuntimeError("Error: Please perform anova\_summary() before multiple comparisons.")  
   
 cols **=** list(sorted(set(self**.**data[by])))  
 descr\_stat **=** self**.**data**.**groupby(by)["Value"]**.**describe()[["count", "mean"]]  
 mse **=** self**.**anova\_table["mean\_sq"]["Residual"]  
 mse\_df **=** self**.**anova\_table["df"]["Residual"]  
 multiple\_comp\_df **=** pd**.**DataFrame(columns**=**["Treatment 1", "Treatment 2", "Mean Diff", "LSD or Omega", "Rejected"])  
 **for** (i, t1) **in** enumerate(cols):  
 **for** (j, t2) **in** enumerate(cols):  
 **if** j **<=** i: **continue**  
   
 **if** method **==** "Fisher":  
 LSD **=** stats**.**t**.**isf(alpha**/**2, mse\_df) **\*** math**.**sqrt(mse **\*** (1**/**descr\_stat["count"][t1] **+** 1**/**descr\_stat["count"][t2]))  
 **if** method **==** "Tukey":  
 k\_treatments **=** len(cols)  
 n\_group **=** stats**.**hmean(descr\_stat["count"]) *# harmonic mean*  
 LSD **=** slb**.**qsturng(1 **-** alpha, k\_treatments, mse\_df) **\*** math**.**sqrt(mse **/** n\_group)  
 mean\_diff **=** descr\_stat["mean"][t1] **-** descr\_stat["mean"][t2]  
 multiple\_comp\_df**.**loc[len(multiple\_comp\_df)] **=** [t1, t2, mean\_diff, LSD, (abs(mean\_diff) **>** LSD)]  
   
 print("="**\***70)  
 print(f"Comparison Method: {method}")  
 print(f"alpha = {alpha:.4f}")  
 *# display(multiple\_comp\_df)*  
 **return** multiple\_comp\_df  
   
   
 **def** anova\_pipeline(self, by): *# CAUTION!!! Always put factor I in index 0.*  
 **if** self**.**method **==** "one-way":  
 self**.**data\_summary(by**=**by)  
 self**.**required\_condition\_summary(by**=**by)  
 self**.**anova\_summary(by**=**by)  
 **elif** self**.**method **==** "rand-block":  
 self**.**data\_summary(by**=**by[0])  
 self**.**required\_condition\_summary(by**=**by[0])  
 self**.**anova\_summary(by**=**by)  
 **else**: *# two-factor*  
 **if** f"{by[0]}\_{by[1]}" **not** **in** self**.**data**.**columns: *# create joint factor column*  
 self**.**data[f"{by[0]}\_{by[1]}"] **=** [f"{f0}\_{f1}" **for** f0, f1 **in** zip(self**.**data[by[0]], self**.**data[by[1]])]  
 self**.**data\_summary(by**=**by[0])  
 self**.**data\_summary(by**=**by[1])  
 self**.**required\_condition\_summary(by**=**f"{by[0]}\_{by[1]}")  
 self**.**anova\_summary(by**=**by)  
   
   
 **def** multiple\_comp\_only\_pipeline(self, by, alpha, method**=**"Tukey"):  
 **if** self**.**anova\_table**.**empty:  
 self**.**anova\_summary(by**=**by, hide\_result**=True**)  
 **return** self**.**multiple\_comp\_summary(by**=**by, alpha**=**alpha, method**=**method)

Data Preprocessing - ANOVA and Multiple Comparison[¶](#gjdgxs)

In [3]:

aqx\_df **=** pd**.**read\_csv("./AQXconcat.csv")

In [4]:

*# fetch central region PM2.5 data*  
*# remove 大城 because of missing data after 2020*  
aqx\_central\_pm25\_df **=** aqx\_df[(aqx\_df**.**region **==** "中區") **&** (aqx\_df**.**itemengname **==** "PM2.5") **&** (aqx\_df**.**sitename **!=** "大城") **&** (aqx\_df**.**year **!=** 2024)]  
display(aqx\_central\_pm25\_df**.**head())

|  | **siteid** | **sitename** | **itemid** | **itemname** | **itemengname** | **itemunit** | **monitordate** | **concentration** | **year** | **month** | **region** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **130405** | 34 | 線西 | 33 | 細懸浮微粒 | PM2.5 | μg/m3 | 2005-06-29 | NaN | 2005 | 6 | 中區 |
| **130408** | 34 | 線西 | 33 | 細懸浮微粒 | PM2.5 | μg/m3 | 2005-06-30 | NaN | 2005 | 6 | 中區 |
| **130430** | 34 | 線西 | 33 | 細懸浮微粒 | PM2.5 | μg/m3 | 2005-07-01 | 40.0 | 2005 | 7 | 中區 |
| **130441** | 34 | 線西 | 33 | 細懸浮微粒 | PM2.5 | μg/m3 | 2005-07-02 | 35.0 | 2005 | 7 | 中區 |
| **130444** | 34 | 線西 | 33 | 細懸浮微粒 | PM2.5 | μg/m3 | 2005-07-03 | 37.0 | 2005 | 7 | 中區 |

In [5]:

*# compute annual mean*  
aqx\_central\_pm25\_annualavg\_df **=** aqx\_central\_pm25\_df**.**groupby(["sitename", "year"])[["concentration"]]**.**mean()  
aqx\_central\_pm25\_annualavg\_df**.**reset\_index(inplace**=True**)  
display(aqx\_central\_pm25\_annualavg\_df**.**head())  
print(len(aqx\_central\_pm25\_annualavg\_df))

|  | **sitename** | **year** | **concentration** |
| --- | --- | --- | --- |
| **0** | 二林 | 2005 | 37.995902 |
| **1** | 二林 | 2006 | 35.971989 |
| **2** | 二林 | 2007 | 39.291545 |
| **3** | 二林 | 2008 | 32.016854 |
| **4** | 二林 | 2009 | 38.367978 |

273

In [6]:

*# drop the years that not all sites have concentration values*  
all\_central\_site\_set **=** set(aqx\_central\_pm25\_annualavg\_df**.**sitename)  
**for** year **in** range(2003, 2025):  
 annual\_df **=** aqx\_central\_pm25\_annualavg\_df[aqx\_central\_pm25\_annualavg\_df**.**year **==** year]  
 annual\_central\_site\_set **=** set(annual\_df**.**sitename)  
 **if** len(all\_central\_site\_set**.**difference(annual\_central\_site\_set)) **!=** 0 **or** year **<** 2018: *# some sites are missing*  
 aqx\_central\_pm25\_annualavg\_df **=** aqx\_central\_pm25\_annualavg\_df[aqx\_central\_pm25\_annualavg\_df**.**year **!=** year] *# remove the year entirely*  
  
display(aqx\_central\_pm25\_annualavg\_df**.**head())  
print(len(aqx\_central\_pm25\_annualavg\_df))  
print(set(aqx\_central\_pm25\_annualavg\_df**.**year))

|  | **sitename** | **year** | **concentration** |
| --- | --- | --- | --- |
| **13** | 二林 | 2018 | 23.354756 |
| **14** | 二林 | 2019 | 20.286908 |
| **15** | 二林 | 2020 | 17.478116 |
| **16** | 二林 | 2021 | 19.139205 |
| **17** | 二林 | 2022 | 15.289773 |

84  
{2018, 2019, 2020, 2021, 2022, 2023}

ANOVA and Multiple Comparison[¶](#30j0zll)

In [7]:

*# ANOVA*  
**import** matplotlib  
matplotlib**.**rc('font', family**=**'Microsoft YaHei') *# Chinese Font ==*  
  
aqx\_central\_pm25\_annualavg\_df**.**rename(columns**=**{"concentration": "Value"}, inplace**=True**)  
  
aqx\_central\_anova **=** myANOVA(data**=**aqx\_central\_pm25\_annualavg\_df, method**=**"rand-block")  
aqx\_central\_anova**.**anova\_pipeline(by**=**["sitename", "year"])

======================================================================  
Descriptive Statistics for All Treatments:

|  | **count** | **mean** | **std** |
| --- | --- | --- | --- |
| **sitename** |  |  |  |
| **二林** | 6.0 | 18.885699 | 2.762864 |
| **南投** | 6.0 | 18.249343 | 2.801826 |
| **嘉義** | 6.0 | 20.397978 | 2.744930 |
| **埔里** | 6.0 | 16.367349 | 3.564038 |
| **大里** | 6.0 | 15.864396 | 1.842102 |
| **崙背** | 6.0 | 19.264300 | 3.865000 |
| **斗六** | 6.0 | 21.040562 | 2.880166 |
| **新港** | 6.0 | 20.234737 | 3.031564 |
| **朴子** | 6.0 | 18.329110 | 2.377903 |
| **竹山** | 6.0 | 20.434350 | 3.797626 |
| **線西** | 6.0 | 16.881643 | 2.175708 |
| **臺西** | 6.0 | 16.585113 | 3.787757 |
| **西屯** | 6.0 | 17.175069 | 2.636505 |
| **豐原** | 6.0 | 15.572016 | 2.523212 |

======================================================================  
Histograms for All Treatments:

Shapiro Test:

|  | **二林** | **南投** | **嘉義** | **埔里** | **大里** | **崙背** | **斗六** | **新港** | **朴子** | **竹山** | **線西** | **臺西** | **西屯** | **豐原** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **shapiro stat** | 0.973407 | 0.996588 | 0.983847 | 0.979244 | 0.94739 | 0.961362 | 0.93254 | 0.939685 | 0.969807 | 0.980155 | 0.968969 | 0.961205 | 0.981867 | 0.920797 |
| **p value** | 0.91449 | 0.999124 | 0.969006 | 0.94774 | 0.719108 | 0.83017 | 0.599852 | 0.656663 | 0.891134 | 0.952324 | 0.885439 | 0.828976 | 0.960453 | 0.511118 |

Bartlett Test:

|  | **All Treatments** |
| --- | --- |
| **bartlett stat** | 5.670967 |
| **p value** | 0.957313 |

======================================================================  
Randomized Block ANOVA Table:

|  | **sum\_sq** | **df** | **mean\_sq** | **F** | **PR(>F)** |
| --- | --- | --- | --- | --- | --- |
| **C(sitename)** | 268.525144 | 13.0 | 20.655780 | 17.156042 | 3.017919e-16 |
| **C(year)** | 541.804864 | 5.0 | 108.360973 | 90.001218 | 7.403576e-28 |
| **Residual** | 78.259643 | 65.0 | 1.203995 | NaN | NaN |
| **Total** | 888.589651 | 83.0 | NaN | NaN | NaN |

In [8]:

*# Multiple Comparisons (Tukey)*  
aqx\_central\_mc\_anova **=** myANOVA(data**=**aqx\_central\_pm25\_annualavg\_df, method**=**"one-way") *# df\_sorted*  
aqx\_central\_mc\_result **=** aqx\_central\_mc\_anova**.**multiple\_comp\_only\_pipeline(by**=**"sitename", alpha**=**.05, method**=**"Tukey")  
aqx\_central\_mc\_result[aqx\_central\_mc\_result**.**Rejected **==** **True**]

======================================================================  
Comparison Method: Tukey  
alpha = 0.0500

Out[8]:

|  | **Treatment 1** | **Treatment 2** | **Mean Diff** | **LSD or Omega** | **Rejected** |
| --- | --- | --- | --- | --- | --- |

In [9]:

*# Multiple Comparisons (Fisher)*  
aqx\_central\_mc\_result **=** aqx\_central\_mc\_anova**.**multiple\_comp\_summary(by**=**"sitename", alpha**=**.05, method**=**"Fisher")  
display(aqx\_central\_mc\_result[aqx\_central\_mc\_result**.**Rejected **==** **True**])

======================================================================  
Comparison Method: Fisher  
alpha = 0.0500

|  | **Treatment 1** | **Treatment 2** | **Mean Diff** | **LSD or Omega** | **Rejected** |
| --- | --- | --- | --- | --- | --- |
| **25** | 嘉義 | 埔里 | 4.030629 | 3.427119 | True |
| **26** | 嘉義 | 大里 | 4.533582 | 3.427119 | True |
| **32** | 嘉義 | 線西 | 3.516335 | 3.427119 | True |
| **33** | 嘉義 | 臺西 | 3.812864 | 3.427119 | True |
| **35** | 嘉義 | 豐原 | 4.825961 | 3.427119 | True |
| **38** | 埔里 | 斗六 | -4.673213 | 3.427119 | True |
| **39** | 埔里 | 新港 | -3.867388 | 3.427119 | True |
| **41** | 埔里 | 竹山 | -4.067001 | 3.427119 | True |
| **47** | 大里 | 斗六 | -5.176166 | 3.427119 | True |
| **48** | 大里 | 新港 | -4.370342 | 3.427119 | True |
| **50** | 大里 | 竹山 | -4.569955 | 3.427119 | True |
| **62** | 崙背 | 豐原 | 3.692283 | 3.427119 | True |
| **66** | 斗六 | 線西 | 4.158919 | 3.427119 | True |
| **67** | 斗六 | 臺西 | 4.455449 | 3.427119 | True |
| **68** | 斗六 | 西屯 | 3.865493 | 3.427119 | True |
| **69** | 斗六 | 豐原 | 5.468546 | 3.427119 | True |
| **73** | 新港 | 臺西 | 3.649624 | 3.427119 | True |
| **75** | 新港 | 豐原 | 4.662721 | 3.427119 | True |
| **81** | 竹山 | 線西 | 3.552707 | 3.427119 | True |
| **82** | 竹山 | 臺西 | 3.849237 | 3.427119 | True |
| **84** | 竹山 | 豐原 | 4.862334 | 3.427119 | True |

In [10]:

*#sorting sites by mean concentration*  
mean\_concentration\_df **=** aqx\_central\_pm25\_annualavg\_df**.**groupby('sitename')['Value']**.**mean()**.**reset\_index()  
mean\_concentration\_df **=** mean\_concentration\_df**.**sort\_values(by**=**'Value', ascending**=False**)  
site\_seq **=** []  
**for** index, row **in** mean\_concentration\_df**.**iterrows():  
 site **=** row['sitename']  
 mean\_concentration **=** row['Value']  
 print(f"{site}: {mean\_concentration}")  
 site\_seq**.**append(site)  
print(site\_seq)  
order **=** range(len(site\_seq))  
order\_df **=** pd**.**DataFrame([order], columns **=** site\_seq)  
display(order\_df)

斗六: 21.04056197974808  
竹山: 20.434350169885004  
嘉義: 20.397977897703786  
新港: 20.234737119332596  
崙背: 19.264299730253423  
二林: 18.885699428626353  
朴子: 18.32910969500441  
南投: 18.24934302612289  
西屯: 17.175068726265753  
線西: 16.88164272125346  
臺西: 16.585113412176227  
埔里: 16.367348724104293  
大里: 15.864395516373255  
豐原: 15.57201640888028  
['斗六', '竹山', '嘉義', '新港', '崙背', '二林', '朴子', '南投', '西屯', '線西', '臺西', '埔里', '大里', '豐原']

|  | **斗六** | **竹山** | **嘉義** | **新港** | **崙背** | **二林** | **朴子** | **南投** | **西屯** | **線西** | **臺西** | **埔里** | **大里** | **豐原** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

In [11]:

*# give relationship for sites*  
consider **=** aqx\_central\_mc\_result[aqx\_central\_mc\_result**.**Rejected **==** **True**]**.**copy()**.**reset\_index()  
relation **=** ["="] **\*** (len(site\_seq)**-**1)  
**for** site **in** site\_seq:  
 closest **=** ''  
 related **=** pd**.**concat([consider[consider['Treatment 1'] **==** site], consider[consider['Treatment 2'] **==** site]])  
 **for** i **in** related**.**index:  
 **if** (related['Treatment 1'][i] **==** site):  
 **if** (closest **==** ''):  
 closest **=** related['Treatment 2'][i]  
 **elif** (order\_df[related['Treatment 2'][i]][0] **<** order\_df[closest][0]):  
 closest **=** related['Treatment 2'][i]  
 **elif** (related['Treatment 2'][i] **==** site):  
 **if** (closest **==** ''):  
 closest **=** related['Treatment 1'][i]  
 **elif** (order\_df[related['Treatment 1'][i]][0] **<** order\_df[closest][0]):  
 closest **=** related['Treatment 1'][i]  
 **else**:  
 print('wrong situation')  
 **if** (closest **!=** ''):  
 no\_need **=** **False**  
 **for** i **in** range(order\_df[site][0], order\_df[closest][0]):  
 **if** (relation[i] **==** '>'):  
 no\_need **=** **True**  
 **break**  
 **if** (**not** no\_need):  
 relation[order\_df[closest][0]**-**1] **=** '>'  
 consider **=** consider**.**drop(related**.**index)  
print(relation)

['=', '=', '=', '=', '=', '=', '=', '>', '=', '=', '=', '=', '=']

Assume that the mean have difference between site 南投 and site 西屯, and divide them into two region for regression.

Regression[¶](#1fob9te)

In [12]:

**import** seaborn **as** sns  
**import** statsmodels.stats.outliers\_influence **as** sso  
**from** statsmodels.tsa.api **import** Holt  
**from** stepwise\_regression **import** step\_reg  
**from** itertools **import** groupby, chain, combinations

In [13]:

**class** MyChi2Test:  
 @staticmethod  
 **def** goodness\_of\_fit(freq\_o, prop\_e, ddof**=**0): *# observed, expected*  
 **if** np**.**sum(prop\_e) **<** 0.9999:  
 warnings**.**warn("Warning: Expected proportions do not add to 1.")  
 sample\_size **=** np**.**sum(freq\_o)  
 freq\_e **=** np**.**array(prop\_e) **\*** sample\_size  
   
 e\_o\_df **=** pd**.**DataFrame({"Observed": freq\_o, "Expected": freq\_e})  
 e\_o\_df **=** e\_o\_df**.**transpose()  
 display(e\_o\_df)  
   
 **if** (freq\_e **<** 5)**.**any():  
 **raise** RuntimeError("Error: Rule of Five is violated.")  
 chi2\_val, p\_val **=** stats**.**chisquare(freq\_o, freq\_e, ddof**=**ddof)  
 chi2\_test\_df **=** pd**.**DataFrame({"Chi2": [chi2\_val], "p value": [round(p\_val, 4)]}, index**=**["Value"])  
 display(chi2\_test\_df)  
   
   
 @staticmethod  
 **def** cont\_table(freq\_o): *# observed*  
 chi2\_val, p\_val, \_, freq\_e **=** stats**.**chi2\_contingency(freq\_o, correction**=False**)  
 print("Expected Frequencies:")  
 display(pd**.**DataFrame(np**.**round(freq\_e, 2), columns**=**freq\_o**.**columns, index**=**freq\_o**.**index))  
   
 **if** (freq\_e **<** 5)**.**any():  
 **raise** RuntimeError("Error: Rule of Five is violated.")  
 chi2\_test\_df **=** pd**.**DataFrame({"Chi2": [chi2\_val], "p value": [round(p\_val, 4)]}, index**=**["Value"])  
 display(chi2\_test\_df)  
   
   
 @staticmethod  
 **def** check\_normality(data, nbins, figsize**=**(9, 5)): *# nbins: number of equal-probability bins*  
 data\_mean, data\_std **=** [np**.**mean(data), np**.**std(data, ddof**=**1)]  
   
 INF **=** 0x3f3f3f3f  
 prop\_e **=** [1 **/** nbins] **\*** nbins  
 bins **=** [**-**INF] **+** [stats**.**norm**.**ppf(i **/** nbins, loc**=**data\_mean, scale**=**data\_std) **for** i **in** range(1, nbins)] **+** [INF]  
   
 freq\_o, \_ **=** np**.**histogram(data, bins) *# counting frequencies of each bin*  
 plt**.**figure(figsize**=**figsize)  
 plt**.**title("Observed Frequencies in Equal Probability Intervals") *# plotting observed freqs*  
 xticks **=** [f"(-INF, {bins[1]:,.2f}]"] **+** [f"[{bins[i]:,.2f}, {bins[i**+**1]:,.2f}]" **for** i **in** range(1, len(bins) **-** 2)] **+** [f"[{bins[len(bins) **-** 2]:,.2f}, INF)"]  
 plt**.**bar(xticks, freq\_o)  
 plt**.**xlabel("Interval")  
 plt**.**ylabel("Frequency")  
 plt**.**axhline(y**=**prop\_e[0]**\***len(data), color**=**'r', label**=**"Expected Frequency")  
 plt**.**legend()  
 plt**.**show()  
   
 MyChi2Test**.**goodness\_of\_fit(freq\_o, prop\_e, ddof**=**2) *# remember ddof = 2*

In [14]:

**class** MyMultipleRegr:  
 **def** \_\_init\_\_(self, data, by, func**=**(**lambda** x: x)): *# by: [y, x\_1, x\_2, ..., x\_k]*  
 self**.**by **=** by  
 self**.**data **=** data  
 self**.**k **=** len(by[1:])  
 self**.**func **=** func  
   
 X\_mat **=** data[by[1:]]  
 X\_mat **=** sm**.**add\_constant(X\_mat) *# add intercept term*  
 y\_vec **=** data[by[0]]**.**apply(func)  
 self**.**regr\_result **=** sm**.**OLS(y\_vec, X\_mat)**.**fit()  
   
 *# standardized residuals & outliers*  
 \_, self**.**regr\_summ\_table, \_ **=** sso**.**summary\_table(self**.**regr\_result, alpha**=**.05)  
 self**.**std\_resid **=** self**.**regr\_summ\_table[:, 10]  
 self**.**outlier\_filter **=** (abs(self**.**std\_resid) **>** 2)  
   
 *# hinge & influential observations*  
 X\_mat **=** X\_mat**.**to\_numpy()  
 H\_mat **=** np**.**dot(X\_mat, np**.**linalg**.**solve(np**.**dot(X\_mat**.**T, X\_mat), X\_mat**.**T))  
 self**.**hinge **=** np**.**diagonal(H\_mat)  
 self**.**influential\_filter **=** (self**.**hinge **>** (3 **\*** (self**.**k **+** 1)) **/** len(self**.**data))  
   
 *# cook's distance & influential observations (does not apply to SLR)*  
 **if** self**.**k **>** 1:  
 y\_resid **=** self**.**regr\_summ\_table[:, 8]  
 mse\_resid **=** self**.**regr\_result**.**mse\_resid  
 self**.**cook\_dist **=** (y\_resid**\*\***2) **/** (self**.**k **-** 1) **/** mse\_resid **\*** self**.**hinge **/** (1 **-** self**.**hinge)**\*\***2  
 self**.**cook\_influential\_filter **=** (self**.**cook\_dist **>** 1)  
   
   
 **def** summary(self):  
 print(self**.**regr\_result**.**summary())  
 print("\n==============================================================================")  
 **if** self**.**k **==** 1:  
 print(f"Testing Correlation Coefficient Between {self**.**by[0]} and {self**.**by[1]}:")  
 corrcoef, p\_val **=** stats**.**pearsonr(self**.**data[self**.**by[1]], self**.**data[self**.**by[0]]**.**apply(self**.**func))  
 cc\_df **=** pd**.**DataFrame(columns**=**["Pearson r", "p value"], index**=**["Value"])  
 cc\_df["Pearson r"]["Value"] **=** round(corrcoef, 4)  
 cc\_df["p value"]["Value"] **=** round(p\_val, 4)  
 display(cc\_df)  
   
 print(f"Mean of {self**.**by[0]} = {self**.**regr\_summ\_table[:, 1]**.**mean():.4f}")  
 print(f"Standard Error of Estimate = {math**.**sqrt(self**.**regr\_result**.**mse\_resid):.4f}")  
   
 print("\nANOVA Table for Regression")  
 anova\_df **=** pd**.**DataFrame(columns**=**["SS", "DF", "MS", "F", "p value"], index**=**["Regression", "Errors", "Total"])  
 anova\_df["F"]["Regression"] **=** self**.**regr\_result**.**fvalue  
 anova\_df["p value"]["Regression"] **=** self**.**regr\_result**.**f\_pvalue  
 anova\_df["MS"]["Regression"] **=** self**.**regr\_result**.**fvalue **\*** self**.**regr\_result**.**mse\_resid  
 anova\_df["DF"]["Regression"] **=** self**.**k  
 anova\_df["MS"]["Errors"] **=** self**.**regr\_result**.**mse\_resid  
 anova\_df["DF"]["Errors"] **=** int(self**.**regr\_result**.**df\_resid)  
 anova\_df["SS"] **=** anova\_df["MS"] **\*** anova\_df["DF"]  
 anova\_df**.**loc["Total"] **=** anova\_df**.**loc["Regression"] **+** anova\_df**.**loc["Errors"]  
 anova\_df["MS"]["Total"] **=** np**.**nan  
 display(anova\_df)  
   
   
 **def** scatter(self): *# also finds outliers and influential obs*  
 print("="**\***70)  
 **for** i, x\_indep **in** enumerate(self**.**by[1:]):  
 **if** i **%** 2 **==** 0: plt**.**figure(figsize**=**(8, 3))  
 ax **=** plt**.**subplot(1, 2, (i **%** 2) **+** 1)  
 plt**.**title(f"{x\_indep} and {self**.**by[0]}")  
 plt**.**scatter(self**.**data[x\_indep], self**.**data[self**.**by[0]], color**=**'gray')  
 plt**.**scatter(self**.**data[x\_indep][self**.**outlier\_filter],   
 self**.**data[self**.**by[0]][self**.**outlier\_filter], label**=**"outlier", color**=**'brown')  
 plt**.**scatter(self**.**data[x\_indep][self**.**influential\_filter],   
 self**.**data[self**.**by[0]][self**.**influential\_filter], label**=**"influential", color**=**'tab:cyan')  
 **if** self**.**k **>** 1:  
 plt**.**scatter(self**.**data[x\_indep][self**.**cook\_influential\_filter],   
 self**.**data[self**.**by[0]][self**.**cook\_influential\_filter], label**=**"cook influential", color**=**'tab:blue')  
 plt**.**xlabel(x\_indep)  
 plt**.**ylabel(self**.**by[0])  
 plt**.**grid()  
 box **=** ax**.**get\_position()  
 ax**.**set\_position([box**.**x0, box**.**y0, box**.**width **\*** 0.9, box**.**height])  
 **if** i **%** 2 **or** x\_indep **==** self**.**by[**-**1]:  
 ax**.**legend(loc**=**'center left', bbox\_to\_anchor**=**(1, 0.5))  
 plt**.**show()  
   
 print("Outliers:")  
 display(pd**.**Series(self**.**std\_resid, name**=**"Standardized Residual")**.**loc[self**.**outlier\_filter])  
 print("\nInfluential Observations:")  
 display(pd**.**Series(self**.**hinge, name**=**"Hinge")**.**loc[self**.**influential\_filter])  
 **if** self**.**k **>** 1:  
 print("\nCook's Influential Observations:")  
 display(pd**.**Series(self**.**cook\_dist, name**=**"Cook's Distance")**.**loc[self**.**cook\_influential\_filter])  
   
   
 **def** slope\_ci(self, index, alpha):  
 print("="**\***50)  
 print(f"Confidence Interval for {self**.**by[index]}'s Coefficient")  
 slope\_ci **=** self**.**regr\_result**.**conf\_int(alpha)**.**loc[self**.**by[index]]  
   
 interval\_df **=** pd**.**DataFrame(columns**=**[f"[{alpha**/**2:.4f}", f"{1 **-** alpha**/**2:.4f}]"], index**=**["confidence"])  
 interval\_df[f"[{alpha**/**2:.4f}"]["confidence"] **=** round(slope\_ci[0], 4)  
 interval\_df[f"{1 **-** alpha**/**2:.4f}]"]["confidence"] **=** round(slope\_ci[1], 4)  
 display(interval\_df)  
   
   
 **def** y\_ci\_pi(self, x\_given, alpha, inverse\_func**=**(**lambda** x: x)):  
 print("="**\***70)  
 print(f"Confidence & Prediction Intervals for x = {x\_given}")  
 pred **=** self**.**regr\_result**.**get\_prediction([1] **+** x\_given)**.**summary\_frame(alpha**=**alpha) *# 1 is intercept*  
   
 interval\_df **=** pd**.**DataFrame(columns**=**[f"[{alpha**/**2:.4f}", f"{1 **-** alpha**/**2:.4f}]"], index**=**["confidence", "prediction"])  
 interval\_df[f"[{alpha**/**2:.4f}"]["confidence"] **=** pred["mean\_ci\_lower"][0]  
 interval\_df[f"[{alpha**/**2:.4f}"]["prediction"] **=** pred["obs\_ci\_lower"][0]  
 interval\_df[f"{1 **-** alpha**/**2:.4f}]"]["confidence"] **=** pred["mean\_ci\_upper"][0]  
 interval\_df[f"{1 **-** alpha**/**2:.4f}]"]["prediction"] **=** pred["obs\_ci\_upper"][0]  
 interval\_df **=** interval\_df**.**applymap(inverse\_func) *# use map() for future versions*  
 interval\_df **=** interval\_df**.**applymap(np**.**round, decimals**=**4)  
 display(interval\_df)  
   
   
 *######### Residual Analysis #########*  
 **def** resid\_normality\_test(self, method**=**"chisquare", nbins**=None**, figsize**=**(9, 5)):  
 print("="**\***70)  
 **if** method **==** "chisquare":  
 print(f"Goodness-of-Fit Test for Standardized Residuals")  
 MyChi2Test**.**check\_normality(self**.**std\_resid, nbins**=**nbins, figsize**=**figsize)  
 **elif** method **==** "shapiro":  
 sm**.**qqplot(self**.**std\_resid, stats**.**norm, fit**=True**, line**=**'45')  
 plt**.**title("Q-Q Plot for Standardized Residuals")  
 plt**.**show()  
   
 sh\_df **=** pd**.**DataFrame(columns**=**["shapiro stat", "p value"], index**=**["Value"])  
 shapiro\_stat, sh\_p\_val **=** stats**.**shapiro(self**.**std\_resid)  
 sh\_df["shapiro stat"]["Value"] **=** shapiro\_stat  
 sh\_df["p value"]["Value"] **=** sh\_p\_val  
 display(sh\_df)  
 **else**:  
 **raise** RuntimeError("Error: Only accepts 'chisquare' or 'shapiro' for method")  
   
   
 **def** residual\_plot(self): *# test for homoscedasticity and outliers*  
 print("="**\***50)  
 print(f"Checking Homoscedasticity with Residual Plot")  
 y\_pred **=** self**.**regr\_summ\_table[:, 2]  
   
 plt**.**title("Standardized Residual Plot")  
 plt**.**scatter(y\_pred, self**.**std\_resid, color**=**'gray')  
 plt**.**scatter(y\_pred[self**.**outlier\_filter], self**.**std\_resid[self**.**outlier\_filter], label**=**"outlier", color**=**'brown')  
 plt**.**axhline(y**=**0, color **=** 'blue')  
 plt**.**axhline(y**=**2, color **=** 'red')  
 plt**.**axhline(y**=-**2, color **=** 'red')  
 plt**.**xlabel(f"Predicted {self**.**by[0]}")  
 plt**.**ylabel("Standardized Residual")  
 plt**.**legend()  
 plt**.**show()  
   
   
 **def** resid\_autocorr\_test(self): *# test for autocorrelation using runs test*  
 print("="**\***50)  
 print(f"Runs Test for Standardized Residuals")  
   
 std\_resid\_median **=** np**.**median(self**.**std\_resid)  
 std\_resid\_ind **=** (self**.**std\_resid **>=** std\_resid\_median)  
 n\_greater **=** np**.**sum(std\_resid\_ind)  
 n\_less **=** (len(self**.**std\_resid) **-** n\_greater)  
 runs **=** len([i **for** i, \_ **in** groupby(std\_resid\_ind)])  
   
 norm\_mean **=** 2**\***n\_greater**\***n\_less **/** (n\_greater **+** n\_less) **+** 1  
 norm\_std **=** (math**.**sqrt(2**\***n\_greater**\***n\_less) **\*** math**.**sqrt((2**\***n\_greater**\***n\_less **-** n\_greater **-** n\_less))  
 **/** math**.**sqrt((n\_greater **+** n\_less)**\*\***2) **/** math**.**sqrt((n\_greater **+** n\_less **-** 1)))  
   
 z\_stat **=** (runs **-** norm\_mean) **/** norm\_std  
 p\_val **=** min(stats**.**norm**.**cdf(z\_stat), 1 **-** stats**.**norm**.**cdf(z\_stat))  
 runs\_df **=** pd**.**DataFrame(columns**=**["# >= Median", "# < Median", "Runs", "Z stat", "p value"], index**=**["Value"])  
 runs\_df["# >= Median"]["Value"] **=** n\_greater  
 runs\_df["# < Median"]["Value"] **=** n\_less  
 runs\_df["Runs"]["Value"] **=** runs  
 runs\_df["Z stat"]["Value"] **=** z\_stat  
 runs\_df["p value"]["Value"] **=** round(p\_val**\***2, 4) *# two-tail*  
 display(runs\_df)  
   
   
 **def** residual\_analysis\_pipeline(self, method**=**"chisquare", nbins**=None**, figsize**=**(9, 5)):  
 self**.**resid\_normality\_test(method, nbins, figsize)  
 self**.**residual\_plot()  
 self**.**resid\_autocorr\_test()  
   
   
 *######### Regression Diagnostics #########*  
 **def** diag\_multicoll(self):  
 print("="**\***50)  
 plt**.**title("Heatmap for Correlation Coefficients")  
 corr\_mat **=** self**.**data[self**.**by]**.**corr()  
 sns**.**heatmap(corr\_mat, annot**=True**)  
 plt**.**show()  
 **for** i **in** self**.**by[1:]:  
 **for** j **in** self**.**by[1:]:  
 **if** i **==** j: **continue**  
 **if** np**.**sum(corr\_mat[i][j] **>** 0.7): *# explicit multicollinearity*  
 warnings**.**warn("Warning: Some correlation coefficients between independent variables have exceeded 0.7...")  
   
 corr\_coef\_df **=** pd**.**DataFrame(columns**=**self**.**by[1:], index**=**["r", "coef"])  
 corr\_coef\_df**.**loc["r"] **=** corr\_mat**.**iloc[0, 1:]  
 corr\_coef\_df**.**loc["coef"] **=** self**.**regr\_result**.**params[1:]  
 print("Comparing correlation coefficients and regression coefficients:")  
 display(corr\_coef\_df)  
 **if** np**.**sum(corr\_coef\_df**.**loc["r"]**\***corr\_coef\_df**.**loc["coef"] **<** 0): *# implicit multicollinearity*  
 warnings**.**warn("Warning: Some signs of r is inconsistent with those of the coefficients...")  
   
   
 **def** diag\_autocorr(self): *# test for autocorrelation using Durbin-Waston*  
 print("="**\***50)  
 plt**.**title("Standardized Residuals Ordered by Sample #")  
 plt**.**plot(self**.**regr\_summ\_table[:, 0], self**.**std\_resid, '-o', color**=**'gray')  
 plt**.**axhline(y**=**0, color**=**'blue')  
 plt**.**xlabel("Sample #")  
 plt**.**ylabel("Standardized Residual")  
 plt**.**show()  
   
 std\_resid\_cut **=** self**.**std\_resid[1:]  
 std\_resid\_rolled **=** np**.**roll(self**.**std\_resid, 1)[1:]  
 dw\_stat **=** np**.**sum((std\_resid\_cut **-** std\_resid\_rolled)**\*\***2) **/** np**.**dot(self**.**std\_resid, self**.**std\_resid)  
 print(f"Durbin-Waston statistic = {dw\_stat:.4f}")  
   
   
 **def** diagnostics\_pipeline(self):  
 self**.**diag\_multicoll()  
 self**.**diag\_autocorr()

In [15]:

**class** MyMBUtilities:  
 @staticmethod  
 **def** sns\_scatter(data, y, x, orders**=**[], logistic**=False**, stratify\_by**=None**, scatter**=True**): *# x is a list*  
 '''  
 :param stratify\_by: the indepedent variable name whose values are used to stratify observations  
 '''  
 **if** stratify\_by:  
 val\_set **=** set(data[stratify\_by])  
 data\_dict **=** {i: data[data[stratify\_by] **==** i] **for** i **in** val\_set}  
   
 **if** len(orders) **==** 0: *# default: linear model*  
 orders **=** [1] **\*** len(x)  
   
 **for** i, x\_indep **in** enumerate(x):  
 **if** stratify\_by:  
 **for** val, stratum **in** data\_dict**.**items():  
 \_ **=** sns**.**regplot(data**=**stratum, x**=**x\_indep, y**=**y, ci**=None**, scatter**=**scatter,  
 order**=**orders[i], logistic**=**logistic,  
 label**=**x\_indep **+** f" ({stratify\_by} = {val})")  
 **else**:  
 \_ **=** sns**.**regplot(data**=**data, x**=**x\_indep, y**=**y, ci**=None**, scatter**=**scatter,  
 order**=**orders[i], logistic**=**logistic,  
 label**=**x\_indep)  
   
 plt**.**title(f"Scatter Plot for {y} and {', '**.**join(x)}")  
 plt**.**xlabel(', '**.**join(x))  
 plt**.**ylabel(y)  
 plt**.**legend()  
 plt**.**grid()  
 plt**.**show()  
   
   
 @staticmethod  
 **def** add\_dummies(data, x): *# x is the name of a qualitative column*  
 dummy\_df **=** pd**.**get\_dummies(data[x], prefix**=**x)**.**astype(float)  
 data **=** pd**.**concat([data, dummy\_df], axis**=**1)  
 **return** data  
   
   
 @staticmethod  
 **def** add\_interaction(data, x1, x2):  
 data[f"{x1}:{x2}"] **=** data[x1] **\*** data[x2]  
 **return** data  
   
   
 @staticmethod  
 **def** add\_previous(data, x, periods):  
 data[f"{x}\_{periods}"] **=** data[x]**.**shift(periods)  
 **return** data**.**dropna()  
   
   
 @staticmethod  
 **def** stepwise(data, by, alpha, direction): *# by = [y, x\_1, ...]*  
 **if** direction **not** **in** ["forward", "backward"]:  
 **raise** RuntimeError("Error: direction should be either 'forward' or 'backward'")  
   
 X\_mat **=** data[by[1:]]  
 X\_mat **=** sm**.**add\_constant(X\_mat) *# add intercept term*  
 y\_vec **=** data[by[0]]  
 **if** direction **==** "forward":  
 result **=** step\_reg**.**forward\_regression(X\_mat, y\_vec, alpha, verbose**=False**)  
 **else**:  
 result **=** step\_reg**.**backward\_regression(X\_mat, y\_vec, alpha, verbose**=False**)  
   
 print(f"{direction**.**capitalize()} Stepwise Regression Result: {result}")  
 **return** result[1:]  
   
   
 @staticmethod  
 **def** best\_subsets(data, by, top\_k**=**5): *# top-k: how many best subsets are returned*  
 by\_powerset **=** list(chain**.**from\_iterable(combinations(by[1:], i) **for** i **in** range(1, len(by[1:])**+**1)))  
 subset\_rsq\_df **=** pd**.**DataFrame(columns**=**["Subset", "Adj. Rsq"])  
   
 **for** by\_subset **in** by\_powerset: *# enumerate all possible subsets*  
 X\_mat **=** data[list(by\_subset)]  
 X\_mat **=** sm**.**add\_constant(X\_mat)  
 y\_vec **=** data[by[0]]  
 adj\_rsq **=** sm**.**OLS(y\_vec, X\_mat)**.**fit()**.**rsquared\_adj  
 subset\_rsq\_df**.**loc[len(subset\_rsq\_df)] **=** [list(by\_subset), adj\_rsq]  
   
 *# sort w.r.t. the adjusted rsquared values*  
 subset\_rsq\_df **=** subset\_rsq\_df**.**sort\_values(by**=**["Adj. Rsq"], ascending**=False**)**.**reset\_index(drop**=True**)  
 print("Best-Subsets Regression Result:")  
 display(subset\_rsq\_df**.**iloc[:top\_k])  
 **return** subset\_rsq\_df["Subset"][:top\_k]

Data Preprocessing - Regression Model for region 1[¶](#3znysh7)

In [31]:

*# including 彰化南部、雲林、嘉義、南投北部*  
central\_region\_1 **=** site\_seq[:8]  
rainfall\_region\_1 **=** ["彰化", "日月潭", "雲林", "嘉義"]  
  
*# including 雲林沿海(臺西)、彰化北部、台中、南投北部*  
central\_region\_2 **=** site\_seq[8:]  
rainfall\_region\_2 **=** ["臺中", "彰化", "日月潭"]  
  
  
print(central\_region\_1)  
print(central\_region\_2)

['斗六', '竹山', '嘉義', '新港', '崙背', '二林', '朴子', '南投']  
['西屯', '線西', '臺西', '埔里', '大里', '豐原']

In [32]:

**from** functools **import** reduce  
  
sites **=** rainfall\_region\_1  
concatenated\_rainfall\_data **=** []  
**for** k, site **in** enumerate(sites):  
 rainfall\_data **=** []  
 **for** i **in** range(107, 113):  
 df **=** pd**.**read\_excel(f'{i}年降雨量概況.ods', engine**=**'odf', header**=None**)  
 search\_result **=** df[df**.**iloc[:, 0]**.**str**.**contains(site, na**=False**)]  
  
 *# Extract data from the row if found*  
 **if** **not** search\_result**.**empty:  
 *# Extract data from columns G to R*  
 extracted\_data **=** search\_result**.**iloc[0, 6:18]**.**reset\_index(drop**=True**)  
 year\_month **=** [f'{i **+** 1911}\_{j}' **for** j **in** range(1, 13)] *# Adjust the year by adding 1911*  
 rainfall\_df **=** pd**.**DataFrame({'year\_month': year\_month, site: extracted\_data})  
  
 *# Append the DataFrame to the list*  
 rainfall\_data**.**append(rainfall\_df)  
 *# Concatenate all the rainfall data into a single DataFrame*  
 concatenated\_rainfall\_data**.**append(pd**.**concat(rainfall\_data, ignore\_index**=True**))  
  
df\_merged\_1 **=** reduce(**lambda** left, right: pd**.**merge(left, right, on**=**'year\_month'), concatenated\_rainfall\_data)  
df\_merged\_1 **=** df\_merged\_1**.**replace('T', 0)  
df\_filtered\_1 **=** df\_merged\_1[df\_merged\_1 **!=** 'X']  
df\_filtered\_1['mean'] **=** df\_filtered\_1[sites]**.**mean(axis**=**1)  
display(df\_filtered\_1**.**head())  
display(df\_filtered\_1**.**tail())  
  
result\_1 **=** df\_filtered\_1[["year\_month", "mean"]]**.**reset\_index(drop**=True**)  
display(result\_1)

|  | **year\_month** | **彰化** | **日月潭** | **雲林** | **嘉義** | **mean** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 2018\_1 | 105.5 | 150.0 | 90.0 | 76.5 | 105.50 |
| **1** | 2018\_2 | 32.0 | 72.0 | 38.5 | 23.7 | 41.55 |
| **2** | 2018\_3 | 31.5 | 45.0 | 37.0 | 31.5 | 36.25 |
| **3** | 2018\_4 | 32.0 | 108.0 | 26.0 | 7.8 | 43.45 |
| **4** | 2018\_5 | 86.0 | 95.0 | 34.5 | 46.5 | 65.50 |

|  | **year\_month** | **彰化** | **日月潭** | **雲林** | **嘉義** | **mean** |
| --- | --- | --- | --- | --- | --- | --- |
| **67** | 2023\_8 | 286.0 | 535.0 | 177.5 | 204.0 | 300.625 |
| **68** | 2023\_9 | 396.0 | 337.5 | 240.5 | 426.0 | 350.000 |
| **69** | 2023\_10 | 37.5 | 50.0 | 75.5 | 45.5 | 52.125 |
| **70** | 2023\_11 | 1.5 | 2.5 | 0.0 | 0.0 | 1.000 |
| **71** | 2023\_12 | 14.0 | 54.5 | 14.5 | 11.5 | 23.625 |

|  | **year\_month** | **mean** |
| --- | --- | --- |
| **0** | 2018\_1 | 105.500 |
| **1** | 2018\_2 | 41.550 |
| **2** | 2018\_3 | 36.250 |
| **3** | 2018\_4 | 43.450 |
| **4** | 2018\_5 | 65.500 |
| **...** | ... | ... |
| **67** | 2023\_8 | 300.625 |
| **68** | 2023\_9 | 350.000 |
| **69** | 2023\_10 | 52.125 |
| **70** | 2023\_11 | 1.000 |
| **71** | 2023\_12 | 23.625 |

72 rows × 2 columns

In [18]:

*# Define the list of sitenames to filter*  
sitenames **=** central\_region\_1  
result **=** result\_1  
  
tem **=** aqx\_df[(aqx\_df['sitename']**.**isin(sitenames))]  
tem **=** tem**.**groupby(["itemname", 'year', 'month'])['concentration']**.**mean()**.**reset\_index()  
tem**.**rename(columns**=**{'concentration': 'concentration\_month\_avg'}, inplace**=True**)  
tem['year\_month'] **=** tem['year']**.**astype(str) **+** '\_' **+** tem['month']**.**astype(str)  
print(tem)  
  
temperature **=** tem[tem["itemname"] **==** "溫度"]  
wind **=** tem[tem["itemname"] **==** "小時風速值"]  
pm25 **=** tem[tem["itemname"] **==** "細懸浮微粒"]  
  
set1 **=** set(temperature["year\_month"])  
set2 **=** set(wind["year\_month"])  
set3 **=** set(pm25["year\_month"])  
set4 **=** set(df\_filtered\_1["year\_month"])  
intersection\_set **=** set1**.**intersection(set2)**.**intersection(set3)**.**intersection(set4)  
  
temperature\_filtered **=** temperature[temperature['year\_month']**.**isin(intersection\_set)]  
temperature\_filtered**.**reset\_index(drop**=True**, inplace**=True**)  
wind\_filtered **=** wind[wind['year\_month']**.**isin(intersection\_set)]  
wind\_filtered**.**reset\_index(drop**=True**, inplace**=True**)  
pm25\_filtered **=** pm25[pm25['year\_month']**.**isin(intersection\_set)]  
pm25\_filtered**.**reset\_index(drop**=True**, inplace**=True**)  
rainfall\_filtered **=** result[result['year\_month']**.**isin(intersection\_set)]  
rainfall\_filtered**.**reset\_index(drop**=True**, inplace**=True**)  
  
df **=** pm25\_filtered[["year\_month",'concentration\_month\_avg']]**.**copy()  
df**.**rename(columns**=**{'concentration\_month\_avg': 'pm25'}, inplace**=True**)  
df["temperature"] **=** temperature\_filtered["concentration\_month\_avg"]  
df["wind"] **=** wind\_filtered["concentration\_month\_avg"]  
df["rainfall"] **=** rainfall\_filtered["mean"]  
df["time"] **=** range(len(df))  
  
*# Convert "rainfall" column to numeric, coerce non-numeric values to NaN*  
df['rainfall'] **=** pd**.**to\_numeric(df['rainfall'], errors**=**'coerce')  
df**.**reset\_index(drop**=True**, inplace**=True**)  
display(df)  
  
df\_1 **=** df

itemname year month concentration\_month\_avg year\_month  
0 一氧化氮 2003 1 6.429739 2003\_1  
1 一氧化氮 2003 2 3.798083 2003\_2  
2 一氧化氮 2003 3 3.806278 2003\_3  
3 一氧化氮 2003 4 3.734340 2003\_4  
4 一氧化氮 2003 5 2.943616 2003\_5  
... ... ... ... ... ...  
4475 風速 2023 12 1.862903 2023\_12  
4476 風速 2024 1 1.879032 2024\_1  
4477 風速 2024 2 1.823276 2024\_2  
4478 風速 2024 3 1.811290 2024\_3  
4479 風速 2024 4 1.560833 2024\_4  
  
[4480 rows x 5 columns]

|  | **year\_month** | **pm25** | **temperature** | **wind** | **rainfall** | **time** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 2018\_1 | 27.681070 | 17.403033 | 2.041803 | 105.500 | 0 |
| **1** | 2018\_2 | 34.125000 | 16.873482 | 2.145536 | 41.550 | 1 |
| **2** | 2018\_3 | 33.882114 | 21.277935 | 1.718057 | 36.250 | 2 |
| **3** | 2018\_4 | 30.506329 | 24.886807 | 1.556050 | 43.450 | 3 |
| **4** | 2018\_5 | 19.261224 | 28.728790 | 1.623750 | 65.500 | 4 |
| **...** | ... | ... | ... | ... | ... | ... |
| **67** | 2023\_8 | 8.201255 | 29.441129 | 1.263710 | 300.625 | 67 |
| **68** | 2023\_9 | 13.295397 | 28.791250 | 0.955417 | 350.000 | 68 |
| **69** | 2023\_10 | 19.827660 | 26.620339 | 1.389407 | 52.125 | 69 |
| **70** | 2023\_11 | 22.503571 | 24.265476 | 1.210714 | 1.000 | 70 |
| **71** | 2023\_12 | 19.897166 | 20.175403 | 1.544355 | 23.625 | 71 |

72 rows × 6 columns

In [19]:

**def** scatter(df, y, x):  
 y\_var **=** df[y]**.**values  
 **for** x\_xs **in** x:  
 plt**.**figure(figsize**=**(4, 3))  
 \_ **=** sns**.**regplot(x **=** df[x\_xs], y **=** y\_var, data **=** df, color **=** 'b', ci **=** **None**)  
 plt**.**title('Scatter Plot for '**+**x\_xs**+**' to '**+**y)  
 plt**.**xlabel(x\_xs)  
 plt**.**ylabel(y)  
 plt**.**show()

In [20]:

aqx\_site\_df **=** df\_1**.**copy()  
aqx\_site\_df["time"] **=** aqx\_site\_df**.**index  
aqx\_site\_df**.**rainfall **=** aqx\_site\_df**.**rainfall**\*\***0.5  
*# aqx\_site\_df.temperature = aqx\_site\_df.temperature\*\*2*  
*# MyMBUtilities.add\_previous(aqx\_site\_df, "pm25", 1)*  
aqx\_site\_df **=** aqx\_site\_df**.**dropna()  
display(aqx\_site\_df**.**head())  
aqx\_site\_df\_1 **=** aqx\_site\_df  
  
items **=** MyMBUtilities**.**best\_subsets(aqx\_site\_df, ["pm25", "temperature", "wind", "rainfall", "time"])[1]  
aqx\_site\_regr **=** MyMultipleRegr(aqx\_site\_df, by**=**["pm25"] **+** items)  
aqx\_site\_regr**.**scatter()  
scatter(aqx\_site\_df, "pm25", items) *#####*  
aqx\_site\_regr**.**diagnostics\_pipeline()  
aqx\_site\_regr**.**residual\_analysis\_pipeline(nbins**=**5)  
aqx\_site\_regr**.**summary()

|  | **year\_month** | **pm25** | **temperature** | **wind** | **rainfall** | **time** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 2018\_1 | 27.681070 | 17.403033 | 2.041803 | 10.271319 | 0 |
| **1** | 2018\_2 | 34.125000 | 16.873482 | 2.145536 | 6.445929 | 1 |
| **2** | 2018\_3 | 33.882114 | 21.277935 | 1.718057 | 6.020797 | 2 |
| **3** | 2018\_4 | 30.506329 | 24.886807 | 1.556050 | 6.591661 | 3 |
| **4** | 2018\_5 | 19.261224 | 28.728790 | 1.623750 | 8.093207 | 4 |

Best-Subsets Regression Result:

|  | **Subset** | **Adj. Rsq** |
| --- | --- | --- |
| **0** | [temperature, wind, rainfall, time] | 0.747537 |
| **1** | [temperature, rainfall, time] | 0.704043 |
| **2** | [temperature, wind, time] | 0.642981 |
| **3** | [temperature, rainfall] | 0.618272 |
| **4** | [temperature, wind, rainfall] | 0.612743 |

======================================================================

Outliers:

29 -2.031486  
37 2.442033  
38 2.122059  
50 2.051037  
59 -2.651621  
63 2.277077  
Name: Standardized Residual, dtype: float64

Influential Observations:

Series([], Name: Hinge, dtype: float64)

Cook's Influential Observations:

Series([], Name: Cook's Distance, dtype: float64)

==================================================

Comparing correlation coefficients and regression coefficients:

|  | **temperature** | **rainfall** | **time** |
| --- | --- | --- | --- |
| **r** | -0.737126 | -0.683807 | -0.321657 |
| **coef** | -0.88008 | -0.464142 | -0.113664 |

==================================================

Durbin-Waston statistic = 1.2149  
======================================================================  
Goodness-of-Fit Test for Standardized Residuals

|  | **0** | **1** | **2** | **3** | **4** |
| --- | --- | --- | --- | --- | --- |
| **Observed** | 13.0 | 15.0 | 18.0 | 14.0 | 12.0 |
| **Expected** | 14.4 | 14.4 | 14.4 | 14.4 | 14.4 |

|  | **Chi2** | **p value** |
| --- | --- | --- |
| **Value** | 1.472222 | 0.479 |

==================================================  
Checking Homoscedasticity with Residual Plot

==================================================  
Runs Test for Standardized Residuals

|  | **# >= Median** | **# < Median** | **Runs** | **Z stat** | **p value** |
| --- | --- | --- | --- | --- | --- |
| **Value** | 36 | 36 | 29 | -1.899039 | 0.0576 |

OLS Regression Results   
==============================================================================  
Dep. Variable: pm25 R-squared: 0.717  
Model: OLS Adj. R-squared: 0.704  
Method: Least Squares F-statistic: 57.30  
Date: Tue, 11 Jun 2024 Prob (F-statistic): 1.37e-18  
Time: 13:32:18 Log-Likelihood: -206.01  
No. Observations: 72 AIC: 420.0  
Df Residuals: 68 BIC: 429.1  
Df Model: 3   
Covariance Type: nonrobust   
===============================================================================  
 coef std err t P>|t| [0.025 0.975]  
-------------------------------------------------------------------------------  
const 49.7533 3.346 14.871 0.000 43.077 56.430  
temperature -0.8801 0.153 -5.752 0.000 -1.185 -0.575  
rainfall -0.4641 0.096 -4.855 0.000 -0.655 -0.273  
time -0.1137 0.025 -4.582 0.000 -0.163 -0.064  
==============================================================================  
Omnibus: 1.185 Durbin-Watson: 1.208  
Prob(Omnibus): 0.553 Jarque-Bera (JB): 0.600  
Skew: 0.145 Prob(JB): 0.741  
Kurtosis: 3.341 Cond. No. 311.  
==============================================================================  
  
Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.  
  
==============================================================================  
Mean of pm25 = 19.5756  
Standard Error of Estimate = 4.3533  
  
ANOVA Table for Regression

|  | **SS** | **DF** | **MS** | **F** | **p value** |
| --- | --- | --- | --- | --- | --- |
| **Regression** | 3257.700684 | 3 | 1085.900228 | 57.299903 | 0.0 |
| **Errors** | 1288.679584 | 68 | 18.95117 | NaN | NaN |
| **Total** | 4546.380267 | 71 | NaN | NaN | NaN |

Data Preprocessing - Regression Model for region 2[¶](#2et92p0)

In [33]:

sites **=** rainfall\_region\_2  
concatenated\_rainfall\_data **=** []  
**for** k, site **in** enumerate(sites):  
 rainfall\_data **=** []  
 **for** i **in** range(107, 113):  
 df **=** pd**.**read\_excel(f'{i}年降雨量概況.ods', engine**=**'odf', header**=None**)  
 search\_result **=** df[df**.**iloc[:, 0]**.**str**.**contains(site, na**=False**)]  
  
 *# Extract data from the row if found*  
 **if** **not** search\_result**.**empty:  
 *# Extract data from columns G to R*  
 extracted\_data **=** search\_result**.**iloc[0, 6:18]**.**reset\_index(drop**=True**)  
 year\_month **=** [f'{i **+** 1911}\_{j}' **for** j **in** range(1, 13)] *# Adjust the year by adding 1911*  
 rainfall\_df **=** pd**.**DataFrame({'year\_month': year\_month, site: extracted\_data})  
  
 *# Append the DataFrame to the list*  
 rainfall\_data**.**append(rainfall\_df)  
 *# Concatenate all the rainfall data into a single DataFrame*  
 concatenated\_rainfall\_data**.**append(pd**.**concat(rainfall\_data, ignore\_index**=True**))  
  
df\_merged\_2 **=** reduce(**lambda** left, right: pd**.**merge(left, right, on**=**'year\_month'), concatenated\_rainfall\_data)  
df\_merged\_2 **=** df\_merged\_2**.**replace('T', 0)  
df\_filtered\_2 **=** df\_merged\_2[df\_merged\_2 **!=** 'X']  
df\_filtered\_2['mean'] **=** df\_filtered\_2[sites]**.**mean(axis**=**1)  
display(df\_filtered\_2**.**head())  
display(df\_filtered\_2**.**tail())  
  
result\_2 **=** df\_filtered\_2[["year\_month", "mean"]]**.**reset\_index(drop**=True**)  
display(result\_2)

|  | **year\_month** | **臺中** | **彰化** | **日月潭** | **mean** |
| --- | --- | --- | --- | --- | --- |
| **0** | 2018\_1 | 103.5 | 105.5 | 150.0 | 119.666667 |
| **1** | 2018\_2 | 25.5 | 32.0 | 72.0 | 43.166667 |
| **2** | 2018\_3 | 35.5 | 31.5 | 45.0 | 37.333333 |
| **3** | 2018\_4 | 30.5 | 32.0 | 108.0 | 56.833333 |
| **4** | 2018\_5 | 73.0 | 86.0 | 95.0 | 84.666667 |

|  | **year\_month** | **臺中** | **彰化** | **日月潭** | **mean** |
| --- | --- | --- | --- | --- | --- |
| **67** | 2023\_8 | 307.5 | 286.0 | 535.0 | 376.166667 |
| **68** | 2023\_9 | 261.0 | 396.0 | 337.5 | 331.500000 |
| **69** | 2023\_10 | 60.0 | 37.5 | 50.0 | 49.166667 |
| **70** | 2023\_11 | 1.5 | 1.5 | 2.5 | 1.833333 |
| **71** | 2023\_12 | 12.5 | 14.0 | 54.5 | 27.000000 |

|  | **year\_month** | **mean** |
| --- | --- | --- |
| **0** | 2018\_1 | 119.666667 |
| **1** | 2018\_2 | 43.166667 |
| **2** | 2018\_3 | 37.333333 |
| **3** | 2018\_4 | 56.833333 |
| **4** | 2018\_5 | 84.666667 |
| **...** | ... | ... |
| **67** | 2023\_8 | 376.166667 |
| **68** | 2023\_9 | 331.500000 |
| **69** | 2023\_10 | 49.166667 |
| **70** | 2023\_11 | 1.833333 |
| **71** | 2023\_12 | 27.000000 |

72 rows × 2 columns

In [34]:

*# Define the list of sitenames to filter*  
sitenames **=** central\_region\_2  
result **=** result\_2  
  
tem **=** aqx\_df[(aqx\_df['sitename']**.**isin(sitenames))]  
tem **=** tem**.**groupby(["itemname", 'year', 'month'])['concentration']**.**mean()**.**reset\_index()  
tem**.**rename(columns**=**{'concentration': 'concentration\_month\_avg'}, inplace**=True**)  
tem['year\_month'] **=** tem['year']**.**astype(str) **+** '\_' **+** tem['month']**.**astype(str)  
print(tem)  
  
temperature **=** tem[tem["itemname"] **==** "溫度"]  
wind **=** tem[tem["itemname"] **==** "小時風速值"]  
pm25 **=** tem[tem["itemname"] **==** "細懸浮微粒"]  
  
set1 **=** set(temperature["year\_month"])  
set2 **=** set(wind["year\_month"])  
set3 **=** set(pm25["year\_month"])  
set4 **=** set(df\_filtered\_2["year\_month"])  
intersection\_set **=** set1**.**intersection(set2)**.**intersection(set3)**.**intersection(set4)  
  
temperature\_filtered **=** temperature[temperature['year\_month']**.**isin(intersection\_set)]  
temperature\_filtered**.**reset\_index(drop**=True**, inplace**=True**)  
wind\_filtered **=** wind[wind['year\_month']**.**isin(intersection\_set)]  
wind\_filtered**.**reset\_index(drop**=True**, inplace**=True**)  
pm25\_filtered **=** pm25[pm25['year\_month']**.**isin(intersection\_set)]  
pm25\_filtered**.**reset\_index(drop**=True**, inplace**=True**)  
rainfall\_filtered **=** result[result['year\_month']**.**isin(intersection\_set)]  
rainfall\_filtered**.**reset\_index(drop**=True**, inplace**=True**)  
  
df **=** pm25\_filtered[["year\_month",'concentration\_month\_avg']]**.**copy()  
df**.**rename(columns**=**{'concentration\_month\_avg': 'pm25'}, inplace**=True**)  
df["temperature"] **=** temperature\_filtered["concentration\_month\_avg"]  
df["wind"] **=** wind\_filtered["concentration\_month\_avg"]  
df["rainfall"] **=** rainfall\_filtered["mean"]  
df["time"] **=** range(len(df))  
  
*# Convert "rainfall" column to numeric, coerce non-numeric values to NaN*  
df['rainfall'] **=** pd**.**to\_numeric(df['rainfall'], errors**=**'coerce')  
df**.**reset\_index(drop**=True**, inplace**=True**)  
display(df)  
  
df\_2 **=** df

itemname year month concentration\_month\_avg year\_month  
0 一氧化氮 2003 1 9.233370 2003\_1  
1 一氧化氮 2003 2 6.677215 2003\_2  
2 一氧化氮 2003 3 5.955233 2003\_3  
3 一氧化氮 2003 4 5.314000 2003\_4  
4 一氧化氮 2003 5 4.089382 2003\_5  
... ... ... ... ... ...  
4719 風速 2023 12 2.819892 2023\_12  
4720 風速 2024 1 2.737634 2024\_1  
4721 風速 2024 2 2.544828 2024\_2  
4722 風速 2024 3 2.504301 2024\_3  
4723 風速 2024 4 2.060556 2024\_4  
  
[4724 rows x 5 columns]

|  | **year\_month** | **pm25** | **temperature** | **wind** | **rainfall** | **time** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 2018\_1 | 19.715068 | 17.064795 | 2.242234 | 119.666667 | 0 |
| **1** | 2018\_2 | 24.434524 | 16.285000 | 2.326607 | 43.166667 | 1 |
| **2** | 2018\_3 | 27.816216 | 20.466452 | 1.928817 | 37.333333 | 2 |
| **3** | 2018\_4 | 27.384181 | 23.898389 | 1.781389 | 56.833333 | 3 |
| **4** | 2018\_5 | 17.721311 | 28.018142 | 2.156557 | 84.666667 | 4 |
| **...** | ... | ... | ... | ... | ... | ... |
| **67** | 2023\_8 | 8.914451 | 28.834946 | 1.931183 | 376.166667 | 67 |
| **68** | 2023\_9 | 12.982222 | 28.356667 | 1.290556 | 331.500000 | 68 |
| **69** | 2023\_10 | 15.425843 | 25.992222 | 2.417778 | 49.166667 | 69 |
| **70** | 2023\_11 | 17.619200 | 23.684921 | 2.090476 | 1.833333 | 70 |
| **71** | 2023\_12 | 14.451613 | 19.750538 | 2.547849 | 27.000000 | 71 |

72 rows × 6 columns

In [35]:

aqx\_site\_df **=** df\_2**.**copy()  
aqx\_site\_df["time"] **=** aqx\_site\_df**.**index  
aqx\_site\_df**.**rainfall **=** aqx\_site\_df**.**rainfall**\*\***0.5  
*# aqx\_site\_df.temperature = aqx\_site\_df.temperature\*\*2*  
*# MyMBUtilities.add\_previous(aqx\_site\_df, "pm25", 1)*  
aqx\_site\_df **=** aqx\_site\_df**.**dropna()  
display(aqx\_site\_df**.**head())  
aqx\_site\_df\_2 **=** aqx\_site\_df  
  
items **=** MyMBUtilities**.**best\_subsets(aqx\_site\_df, ["pm25", "temperature", "wind", "rainfall", "time"])[1]  
aqx\_site\_regr **=** MyMultipleRegr(aqx\_site\_df, by**=**["pm25"] **+** items)  
aqx\_site\_regr**.**scatter()  
scatter(aqx\_site\_df, "pm25", items) *#####*  
aqx\_site\_regr**.**diagnostics\_pipeline()  
aqx\_site\_regr**.**residual\_analysis\_pipeline(nbins**=**5)  
aqx\_site\_regr**.**summary()

|  | **year\_month** | **pm25** | **temperature** | **wind** | **rainfall** | **time** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 2018\_1 | 19.715068 | 17.064795 | 2.242234 | 10.939226 | 0 |
| **1** | 2018\_2 | 24.434524 | 16.285000 | 2.326607 | 6.570134 | 1 |
| **2** | 2018\_3 | 27.816216 | 20.466452 | 1.928817 | 6.110101 | 2 |
| **3** | 2018\_4 | 27.384181 | 23.898389 | 1.781389 | 7.538789 | 3 |
| **4** | 2018\_5 | 17.721311 | 28.018142 | 2.156557 | 9.201449 | 4 |

Best-Subsets Regression Result:

|  | **Subset** | **Adj. Rsq** |
| --- | --- | --- |
| **0** | [temperature, wind, rainfall, time] | 0.712910 |
| **1** | [temperature, rainfall, time] | 0.560500 |
| **2** | [temperature, wind, time] | 0.543393 |
| **3** | [wind, rainfall, time] | 0.524405 |
| **4** | [temperature, wind, rainfall] | 0.511595 |

======================================================================

Outliers:

37 2.544204  
38 2.136174  
59 -2.320597  
63 2.320526  
Name: Standardized Residual, dtype: float64

Influential Observations:

Series([], Name: Hinge, dtype: float64)

Cook's Influential Observations:

Series([], Name: Cook's Distance, dtype: float64)

==================================================

Comparing correlation coefficients and regression coefficients:

|  | **temperature** | **rainfall** | **time** |
| --- | --- | --- | --- |
| **r** | -0.618634 | -0.569233 | -0.397251 |
| **coef** | -0.512459 | -0.290339 | -0.100667 |

==================================================

Durbin-Waston statistic = 1.1516  
======================================================================  
Goodness-of-Fit Test for Standardized Residuals

|  | **0** | **1** | **2** | **3** | **4** |
| --- | --- | --- | --- | --- | --- |
| **Observed** | 13.0 | 19.0 | 14.0 | 11.0 | 15.0 |
| **Expected** | 14.4 | 14.4 | 14.4 | 14.4 | 14.4 |

|  | **Chi2** | **p value** |
| --- | --- | --- |
| **Value** | 2.444444 | 0.2946 |

==================================================  
Checking Homoscedasticity with Residual Plot

==================================================  
Runs Test for Standardized Residuals

|  | **# >= Median** | **# < Median** | **Runs** | **Z stat** | **p value** |
| --- | --- | --- | --- | --- | --- |
| **Value** | 36 | 36 | 29 | -1.899039 | 0.0576 |

OLS Regression Results   
==============================================================================  
Dep. Variable: pm25 R-squared: 0.579  
Model: OLS Adj. R-squared: 0.560  
Method: Least Squares F-statistic: 31.18  
Date: Tue, 11 Jun 2024 Prob (F-statistic): 8.55e-13  
Time: 13:33:30 Log-Likelihood: -196.14  
No. Observations: 72 AIC: 400.3  
Df Residuals: 68 BIC: 409.4  
Df Model: 3   
Covariance Type: nonrobust   
===============================================================================  
 coef std err t P>|t| [0.025 0.975]  
-------------------------------------------------------------------------------  
const 35.2463 2.841 12.405 0.000 29.576 40.916  
temperature -0.5125 0.133 -3.865 0.000 -0.777 -0.248  
rainfall -0.2903 0.082 -3.534 0.001 -0.454 -0.126  
time -0.1007 0.022 -4.646 0.000 -0.144 -0.057  
==============================================================================  
Omnibus: 1.687 Durbin-Watson: 1.145  
Prob(Omnibus): 0.430 Jarque-Bera (JB): 1.390  
Skew: 0.340 Prob(JB): 0.499  
Kurtosis: 2.987 Cond. No. 302.  
==============================================================================  
  
Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.  
  
==============================================================================  
Mean of pm25 = 16.4360  
Standard Error of Estimate = 3.7957  
  
ANOVA Table for Regression

|  | **SS** | **DF** | **MS** | **F** | **p value** |
| --- | --- | --- | --- | --- | --- |
| **Regression** | 1347.74638 | 3 | 449.248793 | 31.182401 | 0.0 |
| **Errors** | 979.684582 | 68 | 14.407126 | NaN | NaN |
| **Total** | 2327.430962 | 71 | NaN | NaN | NaN |

Time Series[¶](#tyjcwt)

In [24]:

**def** Time\_Line\_Chart(y\_data, y\_xs, time\_data, y\_names**=**[]):  
 **for** y **in** y\_names:  
 plt**.**plot(time\_data, y\_data[y])  
 plt**.**legend(labels**=**y\_names)  
 plt**.**xlabel('time')  
 plt**.**ylabel(y\_xs)  
 plt**.**title(y\_xs)  
 plt**.**show()

In [25]:

**def** Auto\_Regression(data, y\_xs, x\_xss**=**[], shift**=**1):  
 data\_2 **=** data**.**copy()**.**dropna()  
 y\_shift **=** y\_xs**+**'\_t'**+**str(shift)  
 data\_2[y\_shift] **=** data\_2[y\_xs]**.**shift(periods **=** shift)  
 y\_var **=** data\_2[y\_xs]**.**values  
 x\_var\_1 **=** data\_2[y\_shift]**.**values  
   
 sns**.**regplot(x **=** x\_var\_1, y **=** y\_var, data **=** data\_2, color **=** 'r', ci **=** **None**)  
 plt**.**title('Line Plot for '**+**y\_xs)  
 plt**.**xlabel(y\_shift)  
 plt**.**ylabel(y\_xs)  
 plt**.**show()  
   
 x\_xss\_auto **=** x\_xss**+**[y\_shift]  
 data\_2\_ **=** data\_2**.**copy()**.**dropna()  
 Y **=** data\_2\_[y\_xs]  
 X **=** data\_2\_[x\_xss\_auto]  
 X **=** sm**.**add\_constant(X) *# Adds a constant term to the predictor*  
   
 olsmod **=** sm**.**OLS(Y, X)  
 result\_reg **=** olsmod**.**fit()  
 print(result\_reg**.**summary())  
   
 **return** result\_reg

region 1[¶](#3dy6vkm)

In [26]:

Time\_Line\_Chart(aqx\_site\_df\_1, "pm25", aqx\_site\_df\_1["time"], ["pm25"])

It seems that there is little seasonal effect in this time line chart. So I try to use autoregression to predict the Pm2.5 value in the next month.

In [28]:

aqx\_site\_pm25\_df **=** df\_1[["year\_month", "pm25"]]**.**copy()  
aqx\_site\_pm25\_df **=** aqx\_site\_pm25\_df**.**iloc[:]**.**reset\_index(drop**=True**)  
aqx\_site\_pm25\_df["time"] **=** aqx\_site\_pm25\_df**.**index  
result\_autoreg **=** Auto\_Regression(aqx\_site\_pm25\_df, "pm25", ["pm25", "time"], shift**=**1)

OLS Regression Results   
==============================================================================  
Dep. Variable: pm25 R-squared: 1.000  
Model: OLS Adj. R-squared: 1.000  
Method: Least Squares F-statistic: 1.849e+30  
Date: Tue, 11 Jun 2024 Prob (F-statistic): 0.00  
Time: 13:32:28 Log-Likelihood: 2115.9  
No. Observations: 71 AIC: -4224.  
Df Residuals: 67 BIC: -4215.  
Df Model: 3   
Covariance Type: nonrobust   
==============================================================================  
 coef std err t P>|t| [0.025 0.975]  
------------------------------------------------------------------------------  
const -7.949e-14 1.34e-14 -5.924 0.000 -1.06e-13 -5.27e-14  
pm25 1.0000 6.35e-16 1.57e+15 0.000 1.000 1.000  
time 3.816e-16 1.75e-16 2.179 0.033 3.21e-17 7.31e-16  
pm25\_t1 1.277e-15 6.36e-16 2.008 0.049 7.41e-18 2.55e-15  
==============================================================================  
Omnibus: 2.532 Durbin-Watson: 0.140  
Prob(Omnibus): 0.282 Jarque-Bera (JB): 1.607  
Skew: 0.096 Prob(JB): 0.448  
Kurtosis: 2.289 Cond. No. 191.  
==============================================================================  
  
Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

region 2[¶](#1t3h5sf)

In [29]:

Time\_Line\_Chart(aqx\_site\_df\_2, "pm25", aqx\_site\_df\_2["time"], ["pm25"])

It seems that there is little seasonal effect in this time line chart. So I try to use autoregression to predict the Pm2.5 value in the next month.

In [30]:

aqx\_site\_pm25\_df **=** df\_2[["year\_month", "pm25"]]**.**copy()  
aqx\_site\_pm25\_df **=** aqx\_site\_pm25\_df**.**iloc[36:]**.**reset\_index(drop**=True**)  
aqx\_site\_pm25\_df["time"] **=** aqx\_site\_pm25\_df**.**index  
result\_autoreg **=** Auto\_Regression(aqx\_site\_pm25\_df, "pm25", ["pm25", "time"], shift**=**1)

OLS Regression Results   
==============================================================================  
Dep. Variable: pm25 R-squared: 1.000  
Model: OLS Adj. R-squared: 1.000  
Method: Least Squares F-statistic: 5.436e+30  
Date: Tue, 11 Jun 2024 Prob (F-statistic): 0.00  
Time: 13:32:31 Log-Likelihood: 1087.4  
No. Observations: 35 AIC: -2167.  
Df Residuals: 31 BIC: -2161.  
Df Model: 3   
Covariance Type: nonrobust   
==============================================================================  
 coef std err t P>|t| [0.025 0.975]  
------------------------------------------------------------------------------  
const 1.132e-14 5.49e-15 2.061 0.048 1.17e-16 2.25e-14  
pm25 1.0000 3.41e-16 2.93e+15 0.000 1.000 1.000  
time -3.053e-16 1.43e-16 -2.130 0.041 -5.98e-16 -1.3e-17  
pm25\_t1 9.992e-16 3.37e-16 2.969 0.006 3.13e-16 1.69e-15  
==============================================================================  
Omnibus: 0.808 Durbin-Watson: 0.574  
Prob(Omnibus): 0.668 Jarque-Bera (JB): 0.794  
Skew: -0.327 Prob(JB): 0.672  
Kurtosis: 2.657 Cond. No. 113.  
==============================================================================  
  
Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Time Series[¶](#tyjcwt)

In [30]:

*# class MyTrendAnalysis:*  
*# def \_\_init\_\_(self, data, by):*  
*# self.data = data.copy()*  
*# self.data.loc[len(data)] = np.nan # append an empty row for forecast*  
*# self.by = by*  
   
   
*# # Smoothing Techniques*  
*# def MA(self, window):*  
*# self.data.drop(columns="MA", inplace=True, errors="ignore")*  
*# self.data["MA"] = self.data[self.by].rolling(window).mean().shift(1)*  
   
   
*# def ES(self, alpha):*  
*# self.data.drop(columns="ES", inplace=True, errors="ignore")*  
*# self.data["ES"] = self.data[self.by].ewm(alpha=alpha, adjust=False).mean().shift(1)*  
*# self.data["ES"][0] = self.data[self.by][0] # initial value*  
   
   
*# def EST(self, alpha, beta, est\_0, et\_0):*  
*# self.data.drop(columns="EST", inplace=True, errors="ignore")*  
*# est = Holt(self.data[self.by].dropna(), initialization\_method='known',*  
*# initial\_level=est\_0, initial\_trend=et\_0).fit(*  
*# smoothing\_level=alpha, smoothing\_trend=beta, optimized=False)*  
*# self.data.drop(columns="EST", inplace=True, errors="ignore")*  
*# self.data["EST"] = est.fittedvalues*  
*# self.data["EST"].iloc[-1] = est.forecast(1)*  
   
   
*# def EST\_multiperiod(self, alpha, beta, est\_0, et\_0, period):*  
*# est = Holt(self.data[self.by].dropna(), initialization\_method='known',*  
*# initial\_level=est\_0, initial\_trend=et\_0).fit(*  
*# smoothing\_level=alpha, smoothing\_trend=beta, optimized=False)*  
*# est\_df = pd.DataFrame(columns=[self.by, "EST"])*  
*# est\_df["EST"] = pd.concat([est.fittedvalues, est.forecast(period)])*  
*# est\_df[self.by] = self.data[self.by]*  
   
*# print("\nMultiple Period Forecast with EST:")*  
*# display(est\_df.tail())*  
   
   
*# # Regression, returns MyMultipleRegr object*  
*# def linear\_regression(self):*  
*# self.data["Regression"] = np.nan*  
*# data\_with\_t = self.data[[self.by]].dropna()*  
*# data\_with\_t["Time"] = data\_with\_t.index*  
*# regr = MyMultipleRegr(data=data\_with\_t, by=[self.by, "Time"])*  
   
*# for i in range(len(self.data)):*  
*# self.data["Regression"][i] = regr.regr\_result.predict([1, i])*  
*# return regr*  
   
   
*# def quadratic\_regression(self):*  
*# self.data["Regression"] = np.nan*  
*# data\_with\_t = self.data[[self.by]].dropna()*  
*# data\_with\_t["Time"] = data\_with\_t.index*  
*# data\_with\_t["Time\_sq"] = data\_with\_t["Time"]\*\*2*  
*# regr = MyMultipleRegr(data=data\_with\_t, by=[self.by, "Time", "Time\_sq"])*  
   
*# for i in range(len(self.data)):*  
*# self.data["Regression"][i] = regr.regr\_result.predict([1, i, i\*\*2])*  
*# return regr*  
   
   
*# # Error Metrics*  
*# def error\_metrics(self):*  
*# err\_df = pd.DataFrame(index=["MAD", "MSE", "MAPE (%)"])*  
*# data\_nna = self.data.dropna() # only evaluate mutually available rows*  
   
*# for smth in ["MA", "ES", "EST", "Regression"]:*  
*# if smth not in data\_nna.columns: continue*  
*# err = data\_nna[smth] - data\_nna[self.by]*  
*# err\_df[smth] = np.nan*  
*# err\_df[smth]["MAD"] = np.mean(np.absolute(err))*  
*# err\_df[smth]["MSE"] = np.mean(err\*\*2)*  
*# err\_df[smth]["MAPE (%)"] = np.mean(np.absolute(err) / data\_nna[self.by] \* 100)*  
   
*# err\_df = err\_df.applymap(np.round, decimals=4) # use map() for future versions*  
*# print("\nError Metrics Table:")*  
*# display(err\_df)*  
   
   
*# # Summarize with Dataframe and Line Chart*  
*# def summary(self):*  
*# print("\nDataframe Overview:")*  
*# display(self.data.head())*  
*# display(self.data.tail())*  
   
*# \_, ax = plt.subplots()*  
*# plt.title(f"Line Chart for {self.by}")*  
*# plt.plot(self.data.index, self.data[self.by], label="Real")*  
*# for smth in ["MA", "ES", "EST", "Regression"]:*  
*# if smth not in self.data.columns: continue*  
*# plt.plot(self.data.index, self.data[smth], label=smth)*  
*# plt.xlabel("Time")*  
*# plt.ylabel(self.by)*  
*# plt.grid()*  
*# box = ax.get\_position()*  
*# ax.set\_position([box.x0, box.y0, box.width \* 0.9, box.height])*  
*# ax.legend(loc='center left', bbox\_to\_anchor=(1, 0.5))*  
*# plt.show()*

In [31]:

*# class MySeasonalAnalysis:*  
*# def \_\_init\_\_(self, data, by, season\_len, start\_season=1):*  
*# self.data = data.copy()*  
*# for i in range(season\_len):*  
*# self.data.loc[len(self.data)] = np.nan # append empty rows for forecast*  
*# self.data["Season"] = (self.data.index + start\_season - 1) % season\_len + 1*  
*# self.data["Time"] = self.data.index*  
*# self.by = by*  
*# self.start\_season = start\_season*  
*# self.season\_len = season\_len*  
   
   
*# def CMA(self, normality\_method="shapiro", nbins=None):*  
*# # calculate CMA*  
*# self.data["CMA"] = self.data[self.by].rolling(window=self.season\_len, center=True).mean()*  
*# if self.season\_len % 2 == 0: # need to roll twice if the season length is even*  
*# self.data["CMA"] = self.data["CMA"].rolling(window=2).mean().shift(-1)*  
   
*# # compute seasonal effect and seasonal index*  
*# self.data["CMA\_seasonal\_eff"] = self.data[self.by] / self.data.CMA*  
*# seasonal\_idx = self.data.groupby("Season")[["CMA\_seasonal\_eff"]].mean()*  
*# seasonal\_idx.columns = ["CMA\_seasonal\_index"]*  
*# seasonal\_idx = seasonal\_idx \* self.season\_len / seasonal\_idx.sum()*  
*# assert abs(seasonal\_idx.sum().values - self.season\_len) < 0.000001*  
*# display(seasonal\_idx)*  
   
*# # compute deseasonalized data*  
*# self.data = self.data.merge(seasonal\_idx.reset\_index()).sort\_values("Time").reset\_index(drop=True)*  
*# self.data["CMA\_deseasoned"] = self.data[self.by] / self.data.CMA\_seasonal\_index*  
   
*# # regression*  
*# print("Performing Regression on Deseasonalized Data...")*  
*# cma\_regr = MyMultipleRegr(self.data[["CMA\_deseasoned", "Time"]].dropna(), by=["CMA\_deseasoned", "Time"])*  
*# cma\_regr.residual\_analysis\_pipeline(method=normality\_method, nbins=nbins)*  
*# cma\_regr.diag\_autocorr()*  
*# cma\_regr.summary()*  
   
*# # forecasting*  
*# self.data["CMA\_pred"] = np.nan*  
*# for i in range(len(self.data)):*  
*# self.data.loc[i, "CMA\_pred"] = cma\_regr.regr\_result.predict([1, self.data["Time"][i]]) \* self.data.loc[i, "CMA\_seasonal\_index"]*  
   
*# # plotting*  
*# plt.plot(self.data.Time, self.data[self.by], label="Real")*  
*# plt.plot(self.data.Time, self.data.CMA\_deseasoned, label=f"Deseason")*  
*# plt.plot(self.data.Time, self.data.CMA\_pred, label=f"Predict")*  
*# plt.xlabel("Time")*  
*# plt.ylabel(self.by)*  
*# plt.title(f"{self.season\_len}-CMA Seasonal Analysis of {self.by}")*  
*# plt.legend()*  
*# plt.grid()*  
*# plt.show()*  
   
*# display(self.data[["Time", "CMA\_pred"]].tail(self.season\_len))*  
   
   
*# def LR(self, normality\_method="shapiro", nbins=None):*  
*# # initial linear regression*  
*# init\_regr = MyMultipleRegr(self.data[[self.by, "Time"]].dropna(), by=[self.by, "Time"])*  
*# self.data["Regr"] = np.concatenate((init\_regr.regr\_summ\_table[:, 2], np.array([np.nan]\*self.season\_len)))*  
   
*# # compute seasonal effect and seasonal index*  
*# self.data["LR\_seasonal\_eff"] = self.data[self.by] / self.data.Regr*  
*# seasonal\_idx = self.data.groupby("Season")[["LR\_seasonal\_eff"]].mean()*  
*# seasonal\_idx.columns = ["LR\_seasonal\_index"]*  
*# seasonal\_idx = seasonal\_idx \* self.season\_len / seasonal\_idx.sum()*  
*# assert abs(seasonal\_idx.sum().values - self.season\_len) < 0.000001*  
*# display(seasonal\_idx)*  
   
*# # compute deseasonalized data*  
*# self.data = self.data.merge(seasonal\_idx.reset\_index()).sort\_values("Time").reset\_index(drop=True)*  
*# self.data["LR\_deseasoned"] = self.data[self.by] / self.data.LR\_seasonal\_index*  
   
*# # regression*  
*# print("Performing Regression on Deseasonalized Data...")*  
*# lr\_regr = MyMultipleRegr(self.data[["LR\_deseasoned", "Time"]].dropna(), by=["LR\_deseasoned", "Time"])*  
*# lr\_regr.residual\_analysis\_pipeline(method=normality\_method, nbins=nbins)*  
*# lr\_regr.diag\_autocorr()*  
*# lr\_regr.summary()*  
   
*# # forecasting*  
*# self.data["LR\_pred"] = np.nan*  
*# for i in range(len(self.data)):*  
*# self.data.loc[i, "LR\_pred"] = lr\_regr.regr\_result.predict([1, self.data["Time"][i]]) \* self.data.loc[i, "LR\_seasonal\_index"]*  
   
*# # plotting*  
*# plt.plot(self.data.Time, self.data[self.by], label="Real")*  
*# plt.plot(self.data.Time, self.data.LR\_deseasoned, label=f"Deseason")*  
*# plt.plot(self.data.Time, self.data.LR\_pred, label=f"Predict")*  
*# plt.xlabel("Time")*  
*# plt.ylabel(self.by)*  
*# plt.title(f"{self.season\_len}-LR Seasonal Analysis of {self.by}")*  
*# plt.legend()*  
*# plt.grid()*  
*# plt.show()*  
   
*# display(self.data[["Time", "LR\_pred"]].tail(self.season\_len))*  
   
   
*# def dummy(self, normality\_method="shapiro", nbins=None):*  
*# # dummy regression*  
*# dummy\_df = MyMBUtilities.add\_dummies(self.data, "Season")*  
*# dummy\_regr = MyMultipleRegr(dummy\_df[[self.by, "Time"] + [f"Season\_{i}" for i in range(1, self.season\_len)]].dropna(),*  
*# by = [self.by, "Time"] + [f"Season\_{i}" for i in range(1, self.season\_len)])*  
   
*# dummy\_regr.residual\_analysis\_pipeline(method=normality\_method, nbins=nbins)*  
*# dummy\_regr.diag\_autocorr()*  
*# dummy\_regr.summary()*  
   
*# # forecasting*  
*# self.data["Dummy\_pred"] = np.nan*  
*# for i in range(len(self.data)):*  
*# season\_ind = [0] \* (self.season\_len - 1)*  
*# if self.data.Season[i] != self.season\_len:*  
*# season\_ind[self.data.Season[i] - 1] = 1*  
*# self.data.loc[i, "Dummy\_pred"] = dummy\_regr.regr\_result.predict([1, self.data["Time"][i]] + season\_ind)*  
   
*# # plotting*  
*# plt.plot(self.data.Time, self.data[self.by], label="Real")*  
*# plt.plot(self.data.Time, self.data.Dummy\_pred, label=f"Predict")*  
*# plt.xlabel("Time")*  
*# plt.ylabel(self.by)*  
*# plt.title(f"{self.season\_len}-Dummy Seasonal Analysis of {self.by}")*  
*# plt.legend()*  
*# plt.grid()*  
*# plt.show()*  
   
*# display(self.data[["Time", "Dummy\_pred"]].tail(self.season\_len))*  
   
   
*# # Error Metrics*  
*# def error\_metrics(self):*  
*# err\_df = pd.DataFrame(index=["MAD", "MSE", "MAPE (%)"])*  
*# sa\_list = [self.by] + list(set(["CMA\_pred", "LR\_pred", "Dummy\_pred"]).intersection(set(self.data.columns)))*  
*# data\_nna = self.data[sa\_list].dropna() # only evaluate mutually available rows*  
   
*# for sa in sa\_list[1:]:*  
*# err = data\_nna[sa] - data\_nna[self.by]*  
*# err\_df[sa] = np.nan*  
*# err\_df[sa]["MAD"] = np.mean(np.absolute(err))*  
*# err\_df[sa]["MSE"] = np.mean(err\*\*2)*  
*# err\_df[sa]["MAPE (%)"] = np.mean(np.absolute(err) / data\_nna[self.by] \* 100)*  
   
*# err\_df = err\_df.applymap(np.round, decimals=4) # use map() for future versions*  
*# print("\nError Metrics Table:")*  
*# display(err\_df)*

In [32]:

*# import matplotlib*  
*# matplotlib.rc('font', family='Microsoft YaHei') # Chinese Font ==*  
  
*# aqx\_site\_pm25\_df = df\_1[["year\_month", "pm25"]].copy()*  
*# aqx\_site\_pm25\_df = aqx\_site\_pm25\_df.iloc[36:].reset\_index(drop=True)*  
*# aqx\_site\_pm25\_df["time"] = aqx\_site\_pm25\_df.index*  
  
*# plt.plot(aqx\_site\_pm25\_df.time, aqx\_site\_pm25\_df.pm25)*  
*# plt.xlabel("Time")*  
*# plt.ylabel("PM2.5")*  
*# plt.title(f"PM2.5 Concentration in central region 1")*  
  
*# print("N/A years:")*  
*# print(aqx\_site\_pm25\_df[aqx\_site\_pm25\_df.pm25.isna()])*  
*# print(len(aqx\_site\_pm25\_df))*  
  
*# aqx\_site\_pm25\_df\_1 = aqx\_site\_pm25\_df*

In [33]:

*# aqx\_site\_season\_1 = MySeasonalAnalysis(aqx\_site\_pm25\_df\_1, "pm25", 12)*  
*# aqx\_site\_season\_1.CMA()*

In [34]:

*# aqx\_site\_season\_1.LR()*

In [35]:

*# aqx\_site\_season\_1.dummy()*

In [36]:

*# aqx\_site\_season\_1.error\_metrics()*

In [37]:

*# import matplotlib*  
*# matplotlib.rc('font', family='Microsoft YaHei') # Chinese Font ==*  
  
*# aqx\_site\_pm25\_df = df\_2[["year\_month", "pm25"]].copy()*  
*# aqx\_site\_pm25\_df = aqx\_site\_pm25\_df.iloc[12:].reset\_index(drop=True)*  
*# aqx\_site\_pm25\_df["time"] = aqx\_site\_pm25\_df.index*  
  
*# plt.plot(aqx\_site\_pm25\_df.time, aqx\_site\_pm25\_df.pm25)*  
*# plt.xlabel("Time")*  
*# plt.ylabel("PM2.5")*  
*# plt.title(f"PM2.5 Concentration in central region 2")*  
  
*# print("N/A years:")*  
*# print(aqx\_site\_pm25\_df[aqx\_site\_pm25\_df.pm25.isna()])*  
*# print(len(aqx\_site\_pm25\_df))*  
  
*# aqx\_site\_pm25\_df\_2 = aqx\_site\_pm25\_df*

In [38]:

*# aqx\_site\_season\_2 = MySeasonalAnalysis(aqx\_site\_pm25\_df\_2, "pm25", 12)*  
*# aqx\_site\_season\_2.CMA()*

In [39]:

*# aqx\_site\_season\_2.LR()*

In [40]:

*# aqx\_site\_season\_2.dummy()*

In [41]:

*# aqx\_site\_season\_2.error\_metrics()*