DAT405 Assignment 2 – Group 67

Student Yaochen Rao – (13.5 hrs)
Student Wanqiu Wang – (20 hrs)

September 13, 2022

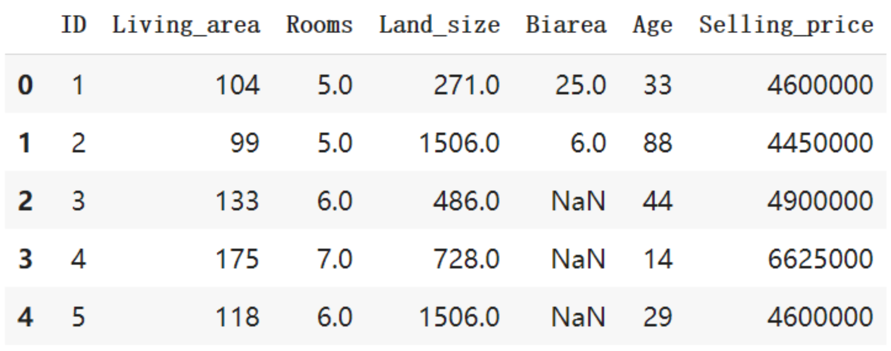
Problem 1

1.Import and Read Dataset

```
#Import package
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('fivethirtyeight')
sns.set()

#Read the dataset
df = pd.read_csv("/content/data_assignment2.csv")
df.head()
```

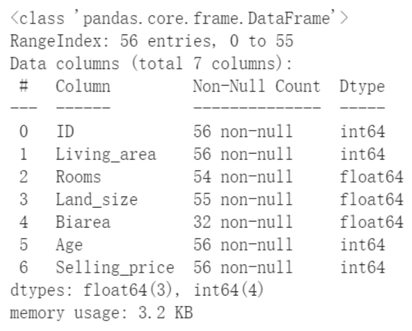


	ID	Living_area	Rooms	Land_size	Biarea	Age	Selling_price
0	1	104	5.0	271.0	25.0	33	4600000
1	2	99	5.0	1506.0	6.0	88	4450000
2	3	133	6.0	486.0	NaN	44	4900000
3	4	175	7.0	728.0	NaN	14	6625000
4	5	118	6.0	1506.0	NaN	29	4600000

Table 1: Result-head of dataset

2.Data Description

```
#Data description
df.info()
```



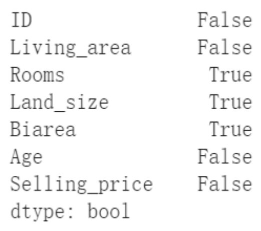
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   ID             56 non-null     int64  
 1   Living_area    56 non-null     int64  
 2   Rooms          54 non-null     float64
 3   Land_size      55 non-null     float64
 4   Biarea         32 non-null     float64
 5   Age            56 non-null     int64  
 6   Selling_price  56 non-null     int64  
dtypes: float64(3), int64(4)
memory usage: 3.2 KB
```

Table 2: Dataset information

We can get that there are no missing values for living_area and selling_price, so we do not do missing value processing.

3.Check Data

```
#check missing value
df.isna().any()
```



```
ID               False
Living_area      False
Rooms             True
Land_size         True
Biarea            True
Age              False
Selling_price    False
dtype: bool
```

Table 3: Check null value

```
#Check duplicate value
duplicated = df.duplicated().sum()
if duplicated:
    print('Duplicated rows in dataset are: {}'.format(duplicated))
else:
    print('Dataset contains no duplicated value.')
```

Dataset contains no duplicated value.

4.Train model

In this part, we trained two models, the first one without training set and test set partitioning, and handled outliers. The second model has test set partitioning, but does not deal with outliers. The purpose of training both models is to explore the effect of whether or not to handle outliers on the model fit, and the effect of dividing the data set on the model fit.

4.1 First Linear Regression Model

```
#Linear correlation coefficient
corr=df['Living_area'].corr(df['Selling_price'],method='pearson')
print(corr)
```

0.5619562093518712
Based on the correlation coefficient of 0.56, we can conclude that there is a correlation between living_area and Selling_price, but the correlation is not strong, indicating that there may be outliers.

```
#Scatter plot
x = df['Living_area']
y = df['Selling_price']
plt.scatter(x, y )
plt.xlabel('living_area (m^2)')
plt.ylabel('selling_price (kr)')
plt.title('Scatter plot of Living_area vs Selling_price')
```

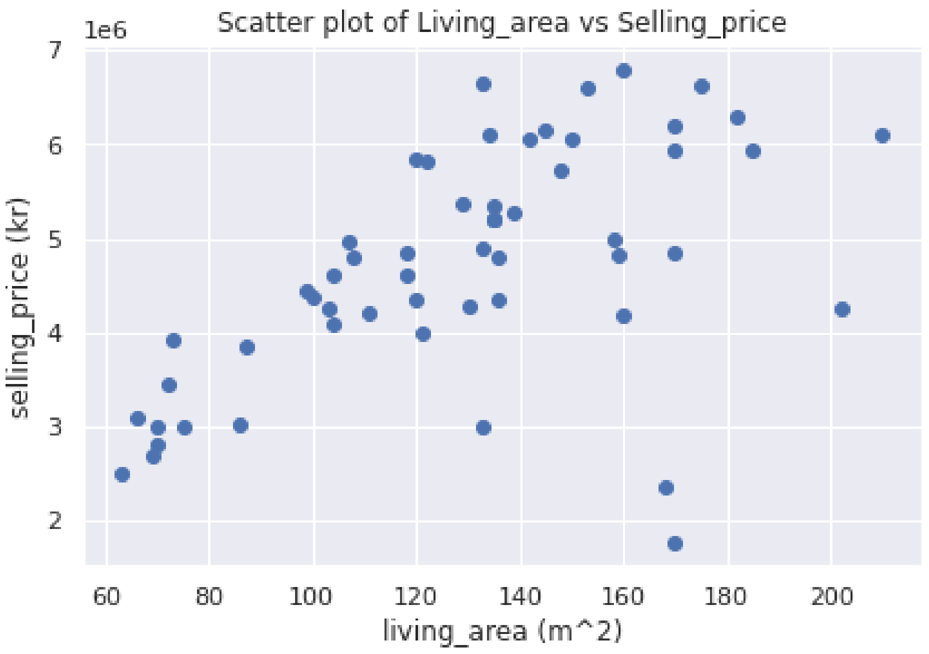


Figure 1: Scatter plot of Living area vs. Selling price

Draw the regression fit line.

```
#Linear Regression Plot
sns.lmplot(x='Living_area',y='Selling_price',data=df,
               legend_out=False,#Render the legend inside the plot frame
               truncate=True,#According to the actual data range, truncate the fitted
    line
               )
plt.xlabel('living_area (m^2)')
plt.ylabel('selling_price (kr)')
plt.title('Linear regression of Living_area vs Selling_price')
```

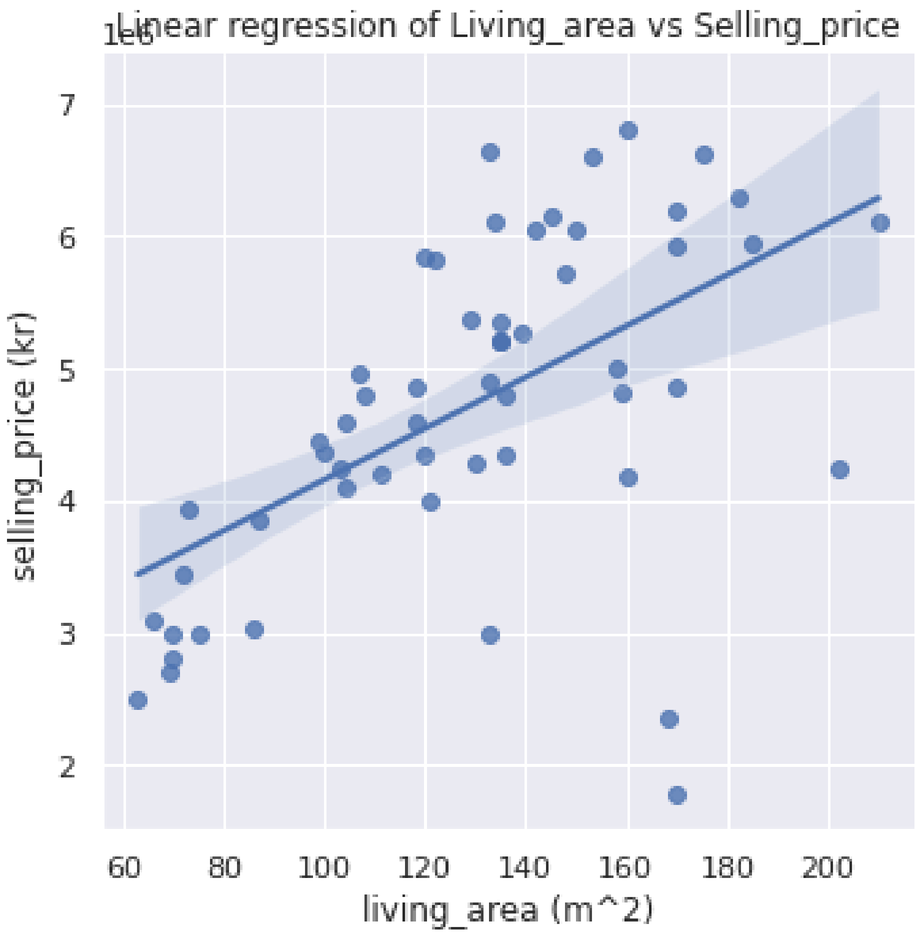


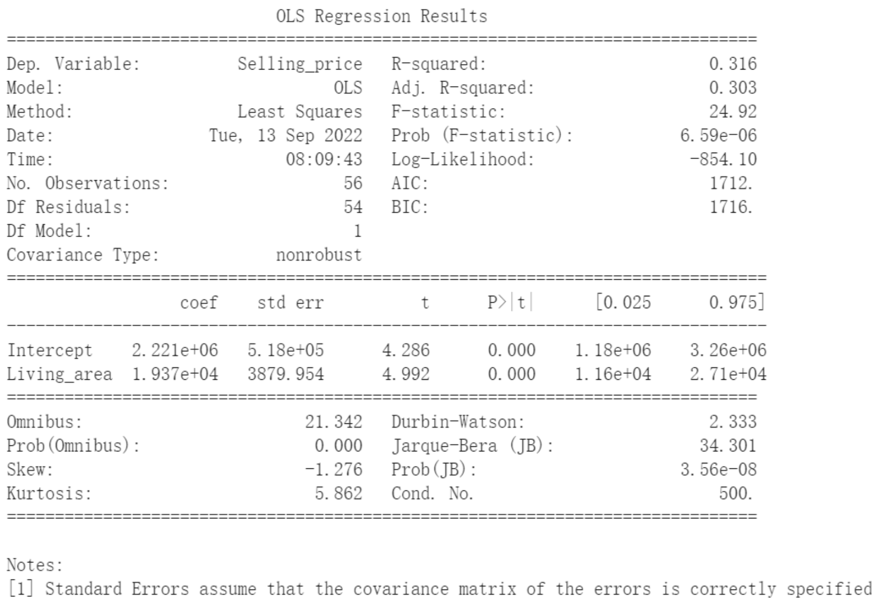
Table 4: Regression plot of living area vs. selling price

According to the scatter plot, we can find that most of the data have certain linear correlation, but there are some outliers. The data is most dense in the middle of the image, and there are some outliers with high area but very low price in the back. Most of the houses have an area of 100-140 square meters and the price is concentrated between 4-6 million kroner. 140-180 square meters are overpriced. 180-200 square meters are under priced.

```
#Solve by OLS Ordinary Least Squares
import statsmodels.api as sm
fit=sm.formula.ols('Selling_price~Living_area',data=df).fit()
print(fit.params)#Intercept: 2.220603e+06,Living_area:1.937014e+04
```

```
Intercept      2.220603e+06
Living area    1.937014e+04
dtype: float64
```

```
#Model Overview
print(fit.summary())
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:          Selling_price   R-squared:                       0.316
Model:                            OLS   Adj. R-squared:                  0.303
Method:                 Least Squares   F-statistic:                     24.92
Date:                Tue, 13 Sep 2022   Prob (F-statistic):           6.59e-06
Time:                        08:09:43   Log-Likelihood:                -854.10
No. Observations:                  56   AIC:                             1712.
Df Residuals:                      54   BIC:                             1716.
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept   2.221e+06   5.18e+05      4.286      0.000    1.18e+06    3.26e+06
Living_area 1.937e+04   3879.954      4.992      0.000    1.16e+04    2.71e+04
==============================================================================
Omnibus:                       21.342   Durbin-Watson:                   2.333
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               34.301
Skew:                          -1.276   Prob(JB):                     3.56e-08
Kurtosis:                       5.862   Cond. No.                         500.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified
```

Table 5: Table of OLS Regression Results

It can be found by the above overview that the value of R-squared is 0.316, which indicates that the accuracy of the obtained model is not high, and the analysis and judgment of outliers should be performed.
a)Answer: The following code handles the outliers. Firstly we run 4 outliers test, then we merge the statistic value with the profit dataset. Next, we calculate the proportion of the number of outliers and use the studentized residual . When the studentized residual is greater than 2 , the corresponding data point is considered to be an outlier. Since the Outliers were identified by studentized residuals, and the proportion of outliers was 3%, also, the proportion of outliers is very small, it can be considered to delete them directly from the data set, thus continuing to model will result in a more robust and reasonable model.

```
#Outlier test
outliers=fit.get_influence()
#High Leverage Value Points (Hat Matrix)
leverage=outliers.hat_matrix_diag
#diffits value
dffits=outliers.dffits[0]
#studentized residuals
resid_stu=outliers.resid_studentized_external
#cook distance
cook=outliers.cooks_distance[0]
```

```
#Combine the statistic values of the above 4 outlier tests
concat_result=pd.concat([pd.Series(leverage,name='leverage'),pd.Series(dffits,name='
    diffits'),pd.Series(resid_stu,name='resid_stu'),pd.Series(cook,name='cook')],axis
    =1)
#Merge the above concat_result result with the profit data set
raw_outliers=pd.concat([df,concat_result],axis=1)
```

```
#Calculate the proportion of the number of outliers and use the studentized residual.
outliers_ratio=sum(np.where(np.abs(raw_outliers.resid_stu)>2,1,0))/raw_outliers.shape
    [0]
print(outliers_ratio)#0.0357
```

0.03571428571428571

```
#Eliminate outliers by screening
none_outliers=raw_outliers.loc[np.abs(raw_outliers.resid_stu)<=2,]
#Applying outlier-free dataset remodeling
fit2=sm.formula.ols('Selling_price~Living_area',data=none_outliers).fit()#Ordinary
    Least Squares
#Overview information
print(fit2.params)#Intercept:1.809821e+06 Living_area:2.359779e+04
print(fit2.summary())#R-squared rise 0.316-->0.540
```

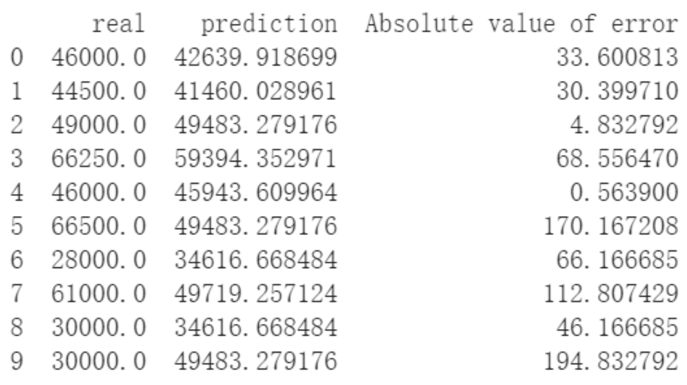
```
Intercept      1.809821e+06
Living_area    2.359779e+04
dtype: float64
                            OLS Regression Results
==============================================================================
Dep. Variable:          Selling_price   R-squared:                       0.540
Model:                            OLS   Adj. R-squared:                  0.531
Method:                 Least Squares   F-statistic:                     61.08
Date:                Tue, 13 Sep 2022   Prob (F-statistic):           2.49e-10
Time:                        08:27:24   Log-Likelihood:                -808.70
No. Observations:                  54   AIC:                             1621.
Df Residuals:                      52   BIC:                             1625.
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept     1.81e+06   3.99e+05      4.540      0.000    1.01e+06    2.61e+06
Living_area   2.36e+04   3019.351      7.816      0.000    1.75e+04    2.97e+04
==============================================================================
Omnibus:                        4.092   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.129   Jarque-Bera (JB):                3.057
Skew:                          -0.462   Prob(JB):                        0.217
Kurtosis:                       3.711   Cond. No.                         492.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Table 6: Table of OLS Regression Results

It can be seen that after removing the outliers (points with studentized residuals is greater than 2), the R2 improves to 0.54, indicating an improvement in the accuracy of the linear fit.
b-1)Answer: For the first linear regression model, the slope is **13957** and the intercept is **1809821**
So the function is y=23597x+1809821

```python
pred=fit2.predict(df[['Living_area']])
#For the comparison of actual and predicted values
df_new=pd.concat([pd.Series(df.Selling_price/100,name='real'),pd.Series(pred/100,name=
    'prediction')],axis=1)
df_new['Absolute value of error']=np.abs((df_new['real']-df_new['prediction'])/100)
print(df_new.head(10))
```



```
      real    prediction  Absolute value of error
0  46000.0  42639.918699                33.600813
1  44500.0  41460.028961                30.399710
2  49000.0  49483.279176                 4.832792
3  66250.0  59394.352971                68.556470
4  46000.0  45943.609964                 0.563900
5  66500.0  49483.279176               170.167208
6  28000.0  34616.668484                66.166685
7  61000.0  49719.257124               112.807429
8  30000.0  34616.668484                46.166685
9  30000.0  49483.279176               194.832792
```

Table 7: Table of real value, prediction value and Absolute value of error

It can be found that there are still significant deviations between the predicted and true values at many points.

d)Answer: Residual plot:

```python
# draw residual plot
sns.residplot(x = "Living_area",
              y = "Selling_price",
              data = df,
              lowess = True)
# show the plot
plt.show()
```

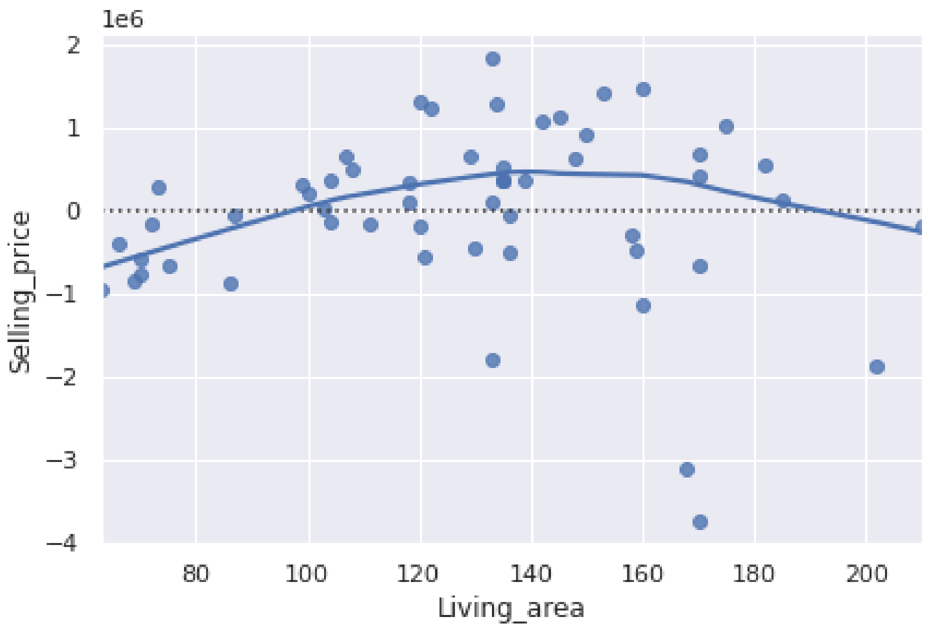


Figure 2: Residual Plot

Comparison on whether to handle outliers or not.

```
plt.figure(figsize=(12,6))
plt.subplot(121)
sns.regplot(x="Living_area", y="Selling_price",data=df,order=1)
plt.xlabel('living_area (m^2)')
plt.ylabel('selling_price (kr)')
plt.title('Linear regression of Living_area vs Selling_price')

plt.subplot(122)
sns.residplot(x="Living_area", y="Selling_price",data=df)
plt.xlabel('living_area (m^2)')
plt.ylabel('residual value (kr)')
plt.title('Residual plot when using Linear regression model')
```

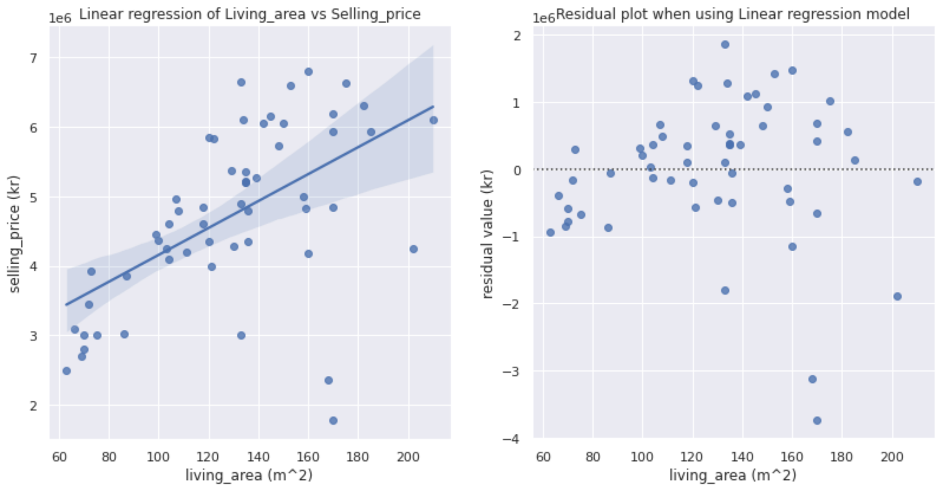


Figure 3: Residual plot when using Linear regression model

```
plt.figure(figsize=(12,6))
plt.subplot(121)
sns.regplot(x="Living_area", y="Selling_price",data=none_outliers,order=1)
plt.title('Regression fit of selling_price records by living_area,order = 1')
plt.xlabel('living_area (m^2)')
plt.ylabel('selling_price (kr)')
plt.title('Linear regression of Living_area vs Selling_price(remove outliers)')

plt.subplot(122)
sns.residplot(x="Living_area", y="Selling_price",data=none_outliers)
plt.xlabel('living_area (m^2)')
plt.ylabel('residual value (kr)')
plt.title('Residual plot when using Linear regression model(remove outliers)')
```

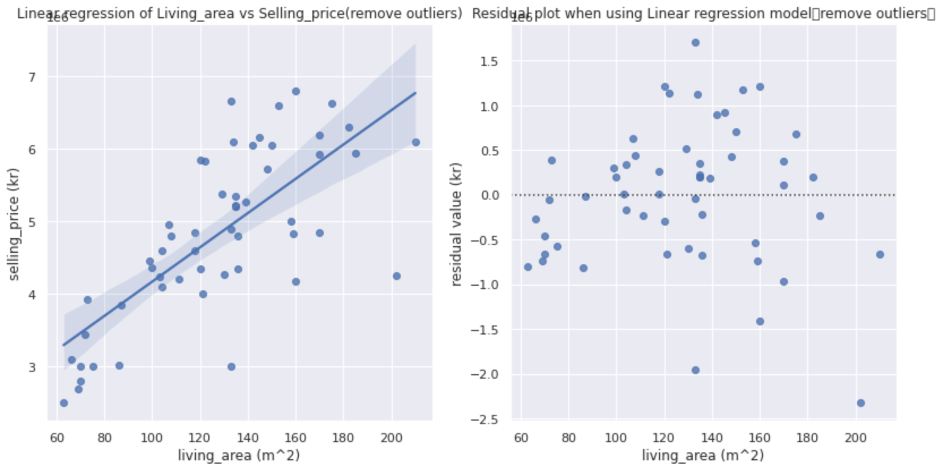


Figure 4: Residual plot when using Linear regression model(remove outliers)

As can be seen from the comparison of the above figure, we have excluded the two outliers in the lower right corner. These two points are[170,1775000] , [168,2360000].The two outliers could be analyzed according to the studentized residuals is greater than 2 and were removed in the above code operation, and the new linear regression equation was obtained with the analysis of the removed data.

```
df1 = [[100],[150],[200]]
df1 = pd.DataFrame(df1,columns=['Living_area'])
y1=fit2.predict(df1['Living_area'])
y1
```

```
0        4.169601e+06
1        5.349490e+06
2        6.529380e+06
dtype: float64
```

c-1)Answer: First linear regression model predict living area are

living_area	prediction value
100	4.169601e+06
150	5.349490e+06
200	6.529380e+06

4.2 Second Linear Regression Model

```
#Separate data into dependent and independent variable
X = np.array(df['Living_area']).reshape(-1, 1)
y = np.array(df['Selling_price']).reshape(-1, 1)
```

```
#Splitting Data into train validation and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20)
```

```
# data standardization with sklearn
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
#Fitting into model
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_train,y_train)
```

```
LinearRegression()
```

```
#Predict on test set
lin_pred = reg.predict(X_test)
```

```
#print the intercept
print(reg.intercept_)
```

[2684877.64475825]

```
#Print the slope
print(reg.coef_)
```

[[15114.23182851]]
b-2)Answer: For the second linear regression model, the slope is **15114** and the intercept is **2684877**. **So the function is y=15114x+2684877**

```
#Draw the plot
y_pred = reg.predict(X_test)
plt.scatter(X_test, y_test, color ='b')
plt.plot(X_test, y_pred, color ='k')
plt.title('')
plt.show()
```

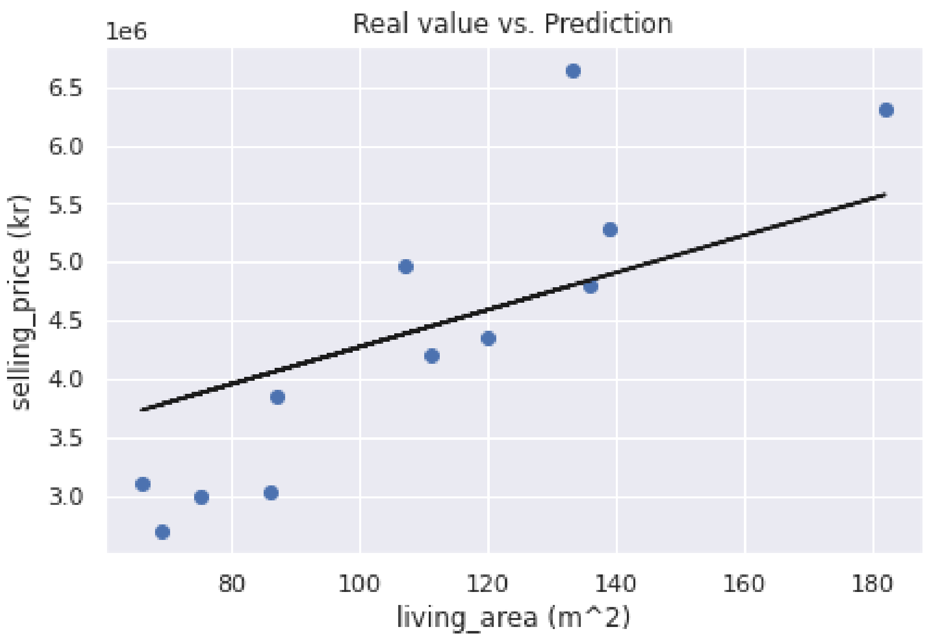


Figure 5: Scatter plot

It can be found that the accuracy is still not high based on the images, and the error between the true and predicted values of each point is still high.

```
p1 = reg.predict([[100]])[0]
p2 = reg.predict([[150]])[0]
p3 = reg.predict([[200]])[0]
print('Prediction when living_area is 100, 150, 200 are', p1, p2, p3)
```

Prediction when living_area is 100, 150, 200 are [4196300.82760943] [4952012.41903502] [5707724.01046061]
c-2)Answer: Second linear regression model predict living area are

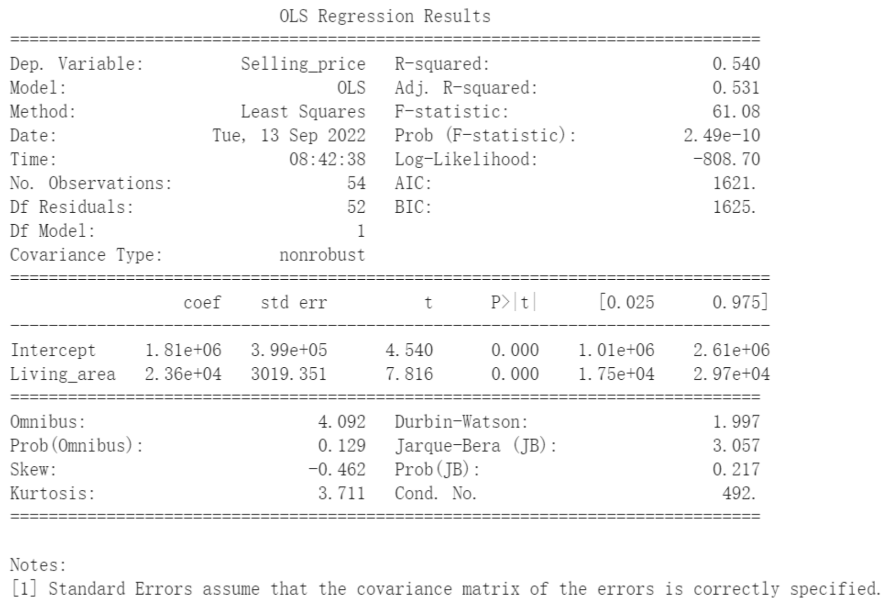
living_area	prediction value
100	4196300
150	4952012
200	5707724

6. Model Evaluation

6.1 First linear regression model evaluation

e-1)Answer: The result of first linear regression model is as follows

```
print(fit2.summary())
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:          Selling_price   R-squared:                       0.540
Model:                            OLS   Adj. R-squared:                  0.531
Method:                 Least Squares   F-statistic:                     61.08
Date:                Tue, 13 Sep 2022   Prob (F-statistic):           2.49e-10
Time:                        08:42:38   Log-Likelihood:                -808.70
No. Observations:                  54   AIC:                             1621.
Df Residuals:                      52   BIC:                             1625.
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept     1.81e+06   3.99e+05      4.540      0.000    1.01e+06    2.61e+06
Living_area   2.36e+04   3019.351      7.816      0.000    1.75e+04    2.97e+04
===============================================================================
Omnibus:                        4.092   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.129   Jarque-Bera (JB):                3.057
Skew:                          -0.462   Prob(JB):                        0.217
Kurtosis:                       3.711   Cond. No.                         492.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Table 8: OLS Regression Results

6.2 Second Linear Regression model Evaluation

e-2)Answer: The result of second linear regression model is as follows
Coefficient of Determination

```
#Using coefficient of determination to judge how well the regression equation fits.
R_squared_lr = reg.score(X_test,y_test,sample_weight=None)
print('Coefficient of determination: %f '%(R_squared_lr))
```

Coefficient of determination: 0.556882
Measure RMSE

```python
#Measure the RMSE of the regression model on the training set
from sklearn.metrics import mean_squared_error
lin_rmse = np.sqrt(mean_squared_error(y_test, lin_pred))
print(lin_rmse)
```

903271.7637222876

Cross Validation

```python
#Use cross validation to better evaluate model
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator = reg, X = X_train, y = y_train,scoring="
    neg_mean_squared_error",cv = 10)
svr_rmse_scores = np.sqrt(-scores)
```

```python
def display_scores(scores):
    print("Scores:",scores)
    print("Mean:",scores.mean())
    print("Standard deviation:",scores.std())
```

```python
display_scores(svr_rmse_scores)
```

Scores: [756557.20320321 836965.10687395 1759120.3761306 611090.13205094 776273.23895808 1414869.12366205 1237272.19166388 302143.03931493 977398.5593123 1586505.47092129]
Mean: 1025819.444209123
Standard deviation: 437797.32091125986

Measure AUC

```python
from sklearn.metrics import explained_variance_score
score = explained_variance_score(y_test,lin_pred)
print(score)
```

0.6390895333470274

e-3)Answer: How to improve the model: 1. Increase the data in the dataset. Since it is house price data, so if the data set is too small it will lead to a big difference in selling_price, resulting in a poor model fit. So if new data is collected and labeled regularly, it will greatly improve the accuracy of the model.2. Remove outliers.3. Perform feature scaling on the data, i.e., normalization.4. Divide the data set according to the training set and the test set.5. Perform k-fold cross validation on the data, divide the training set into a smaller training set and a validator, and train the model based on these smaller training The model is trained and evaluated based on these smaller training sets.

Problem 2

```
#Import and read dataset
from sklearn.datasets import load_iris
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
import pandas as pd
import numpy as np

plt.style.use('fivethirtyeight')
sns.set()
```

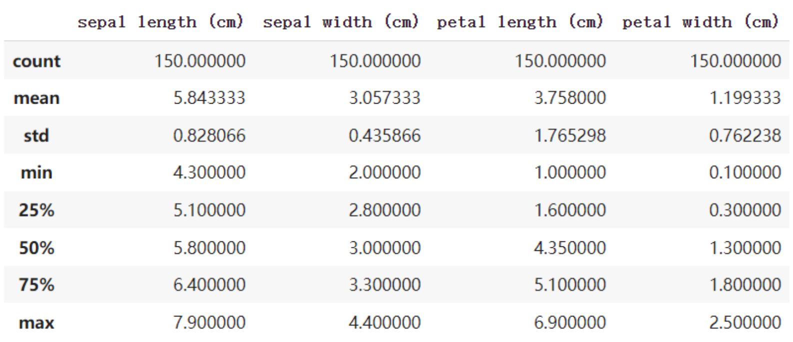
```
data = load_iris()
print(dir(data))
```

['DESCR', 'data', 'data_module', 'feature_names', 'filename', 'frame', 'target', 'target_names']

```
ris_target = data.target #Get the label corresponding to the data
iris_features = pd.DataFrame(data=data.data, columns=data.feature_names) #Convert to
    DataFrame format using Pandas
```

```
#Use the value_counts function to view the number of each category
pd.Series(iris_target).value_counts()
```

```
# View the mean, variance, min, max, quartiles of the data
iris_features.describe()
```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Figure 6: Data description

```
# Merge labels and feature information
iris_copy = iris_features.copy()
iris_copy['target'] = iris_target
```

```
# Scatter visualization of feature and label combinations
# The scatter distribution of different feature combinations for different classes of
    flowers in the 2D case, and the approximate discriminative power.
sns.pairplot(data=iris_copy,diag_kind='hist',hue= 'target')
plt.show()
```

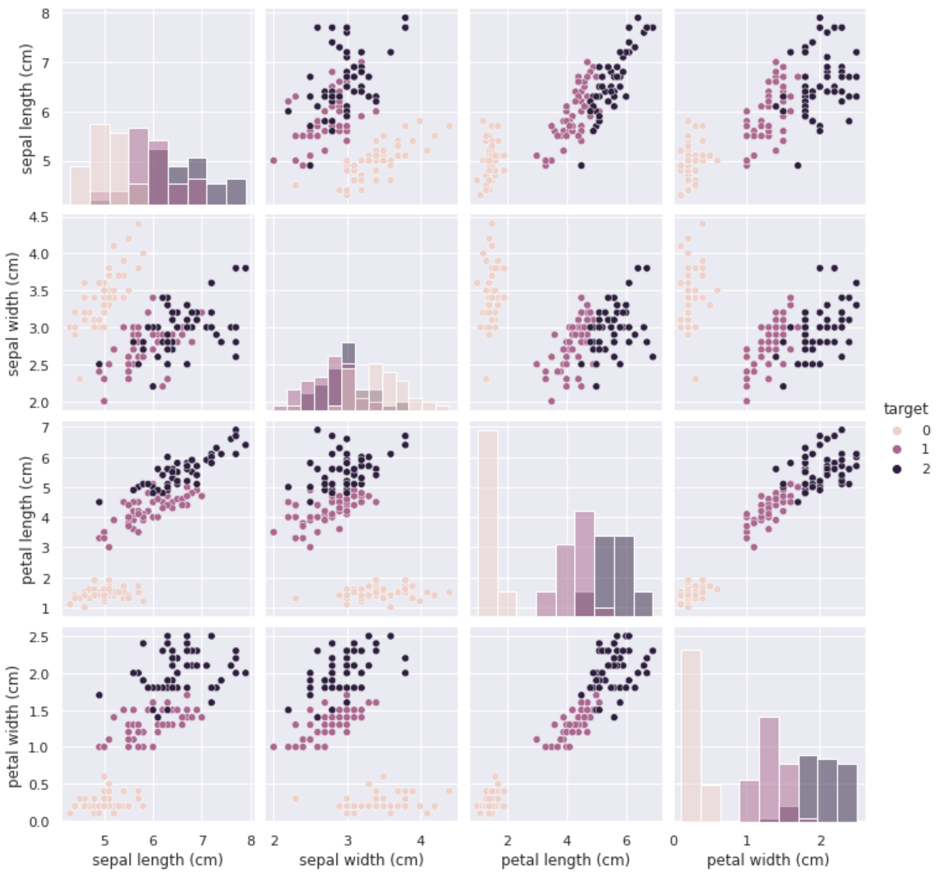


Figure 7: Correlation between data

```
# Divide the data into training set and test set, 20% as test set
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(iris_features,iris_target,test_size
    =0.2,random_state=2020)
```

logistic regression

```
from sklearn.linear_model import LogisticRegression

# Define logistic regression model without activation function
lrm=LogisticRegression(random_state=0,solver='lbfgs')
# Train a logistic regression model on the training set
lrm.fit(X_train,y_train)
```

```
print('the weight of Logistic Regression:\n',lrm.coef_)
print('the intercept(w0) of Logistic Regression:\n',lrm.intercept_)
```

```
the weight of Logistic Regression:
 [[-0.45928925  0.83069893 -2.26606528 -0.99743983]
 [ 0.33117319 -0.72863427 -0.06841147 -0.98711029]
 [ 0.12811605 -0.10206466  2.33447675  1.98455011]]
the intercept(w0) of Logistic Regression:
 [  9.43880648   3.93047366 -13.36928014]
```

```
#Distribute predictions using the trained model on the training and test sets
train_predict=lrm.predict(X_train)
test_predict=lrm.predict(X_test)
```

```
# Model evaluation
lrm.score(X_test,y_test)
```

0.8666666666666667

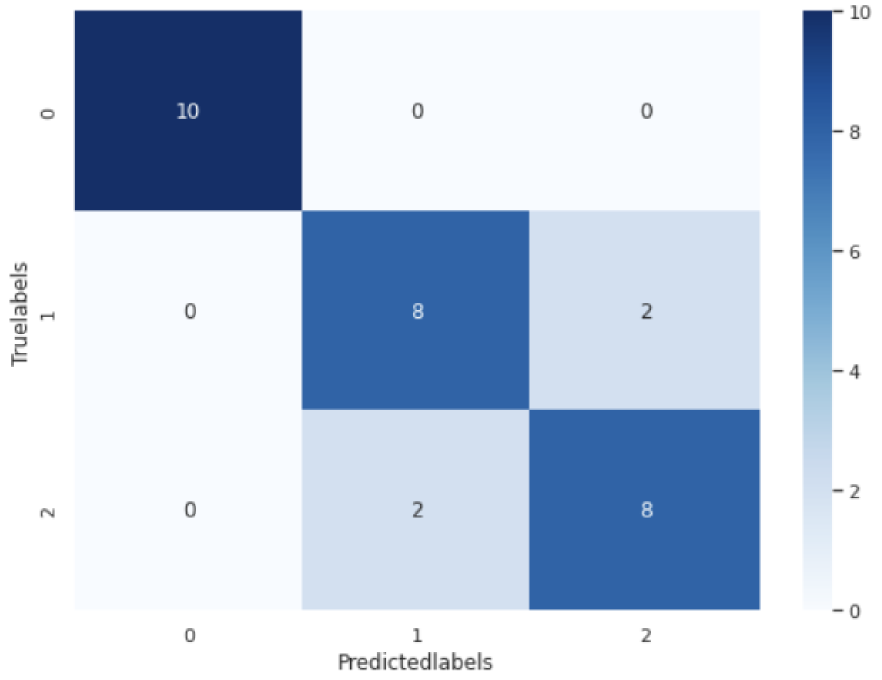
a)Answer:The accuracy of the model on the training and test sets can be calculated, and then the confusion matrix is output for the test set. It can be found from the diagonal of the matrix that most of the classification results are correct. and also it can be seen that there are misclassified examples, but the number is small. It indicates that the classification by logistic regression still has errors and the accuracy still needs to be improved.

```
from sklearn import metrics
##Use accuracy to evaluate the model effect
print('The accuracy of the Logistic Regression is:',metrics.accuracy_score(y_train,
    train_predict))
print('The accuracy of the Logistic Regression is:',metrics.accuracy_score(y_test,
    test_predict))

##View confusion matrix
confusion_matrix_result=metrics.confusion_matrix(y_test,test_predict)
print('The confusion matrix result:\n',confusion_matrix_result)

## Use a heatmap to visualize the results and draw a confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix_result,annot=True,cmap='Blues')
plt.xlabel('Predictedlabels')
plt.ylabel('Truelabels')
plt.show()
```

```
The accuracy of the Logistic Regression is: 0.9833333333333333
The accuracy of the Logistic Regression is: 0.8666666666666667
The confusion matrix result:
 [[10  0  0]
 [ 0  8  2]
 [ 0  2  8]]
```



k-nearest neighbours

```
from sklearn import neighbors, datasets
# Import the dataset
iris=load_iris()
iris_data=iris.data
iris_target = iris.target
```

```
# Divide the data into training set and test set, 20% as test set
x_train,x_test,y_train,y_test= train_test_split(iris_data,iris_target,test_size=0.20)
```

```
# Train KNN using the training set
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5) #using k = 5
knn.fit(x_train,y_train)
y_predict = knn.predict(x_test)
```

```
# Model evaluation
knn.score(x_test,y_test)
```

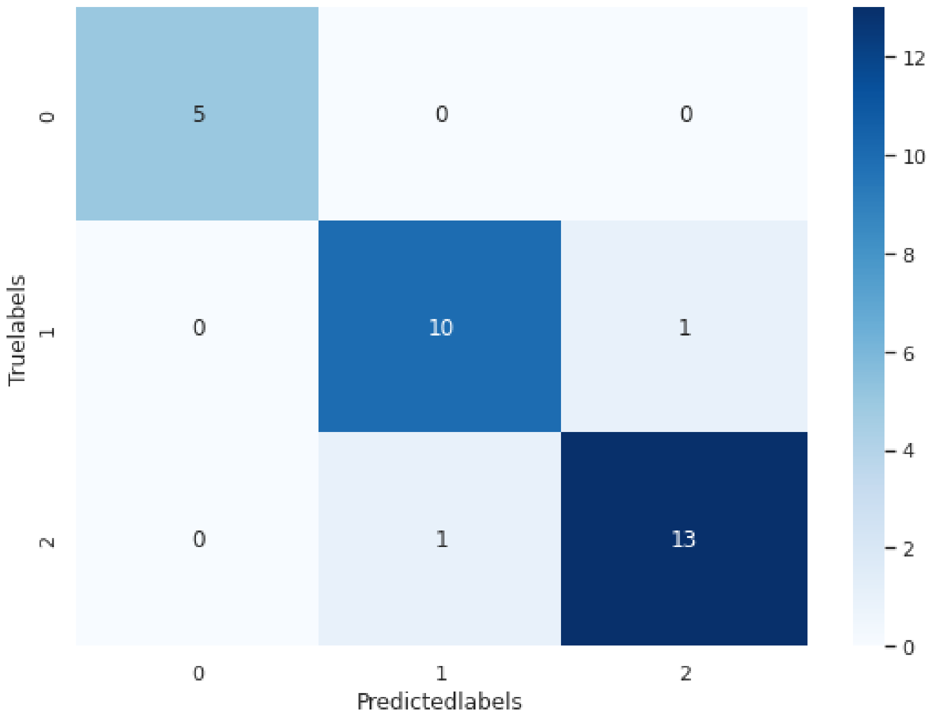
```
0.9333333333333333
```

```
knn.score(x_train,y_train)
```

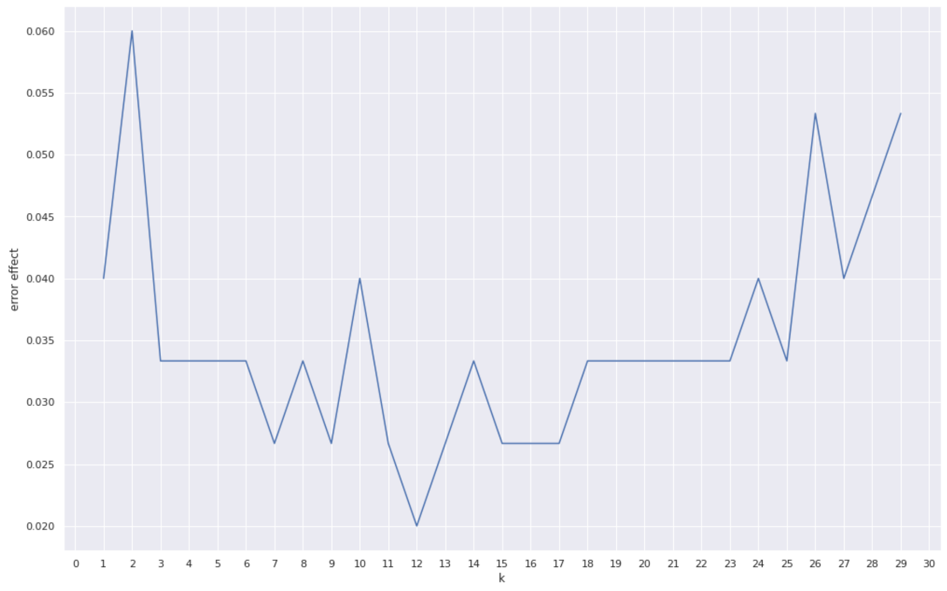
```
0.9916666666666667
```

```
## View confusion matrix
confusion_matrix_result=metrics.confusion_matrix(y_test,y_predict)
print('The confusion matrix result:\n',confusion_matrix_result)

## Use the heat map to visualize the results and draw the confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix_result,annot=True,cmap='Blues')
plt.xlabel('Predictedlabels')
plt.ylabel('Truelabels')
plt.show()
```



```
from matplotlib.pyplot import MultipleLocator
from sklearn.model_selection import cross_val_score
plt.figure(figsize=(15,10))
k_range = range(1,30)
k_error = []
x = iris.data
y = iris.target
#Loop to see the error effect
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    #cv parameters to divide training set and test set
    scores = cross_val_score(knn,x,y,cv=6)
    k_error.append(1-scores.mean())
x_major_locator=MultipleLocator(1)
ax=plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.plot(k_range,k_error)
plt.xlabel('k')
plt.ylabel('error effect')
plt.show()
```



b)Answer: It can be found from the above figure that when the value of k is 12, the error is the smallest and the accuracy of the model is higher. As the value of k increases, the accuracy of the model keeps changing. The highest accuracy will be reached at a point in the middle, and then the accuracy will decrease. If the value of k is selected to be small, over-fitting may occur, which will cause noise to affect the training of the model. However, if the value of k is too large, there will be a phenomenon of just fitting, and the final predicted categories are all the same.

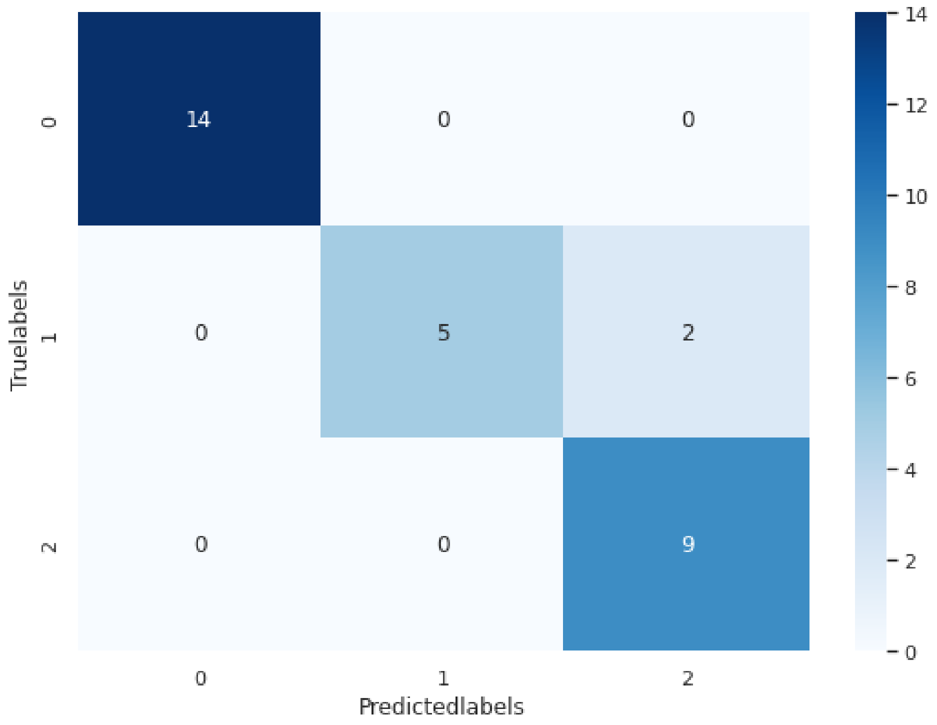
```
# so choose k = 12
iris=load_iris()
iris_data=iris.data
iris_target = iris.target
x_train,x_test,y_train,y_test= train_test_split(iris_data,iris_target,test_size=0.20)
knn2 = KNeighborsClassifier(n_neighbors=12)
knn2.fit(x_train,y_train)
y_predict2 = knn2.predict(x_test)
```

```
# Model evaluation
knn2.score(x_test,y_test)
```

0.9333333333333333

```
confusion_matrix_result=metrics.confusion_matrix(y_test,y_predict2)
print('The confusion matrix result:\n',confusion_matrix_result)

plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix_result,annot=True,cmap='Blues')
plt.xlabel('Predictedlabels')
plt.ylabel('Truelabels')
plt.show()
```



c)Answer: As the picture shows, the confusion matrix is output for the test set. It can be found that the matrices obtained in the case of two different k values are also different. When k=5, there is a misclassification point, but it does not exist when k=12, which means that when k=12, the performance of the model is the highest.And then you can compare the logistic regression confusion matrix with the KNN confusion matrices of k=5 and k=12. It will be found that when k=12, the matrix only has values on the diagonal, which means that all classifications are correct and the model has the highest accuracy. While the other two have misclassified values. However, the accuracy of k=5 is still higher than that of logistic regression, because when k=5, the number of misclassifications is less than that of logistic regression classification.

Problem 3

According to the R_squared comparison between the two linear regression model, the accuracy of the function obtained by linear regression without training set and test set differentiation is very low, and the accuracy of the model obtained after removing the outliers is close to that of the model obtained by training.
I think each set has its own function, for example, the training set is used to train the model, give the model inputs and corresponding outputs, and let the model learn the relationship between them to get more accurate results. The validation set is used to estimate the training level of the model, such as accuracy, prediction error, etc. We can use the validation set to select the best model. The test set is the result of the input data on the final model, which is the output of the trained model on the simulated "new" input data. The test set can only be used at the end to test the performance of the model, not for training.
So it is technically possible to use all the data of a dataset for model fitting, and it is also possible to calculate the actual and predicted values of the model, and thus the error. However, this may have the problem that the obtained model is only applicable to the present dataset. That is, by using a dataset for both training and testing, the performance evaluation metrics of the model are inaccurate and an accurate model cannot be obtained.