Home Work 2 (81)

(APANPS5335_002_2023_3 - MACHINE LEARNING: CONCEPTS & APPLICATION)

Submitted by: Eli Guo

UNI: yg2869

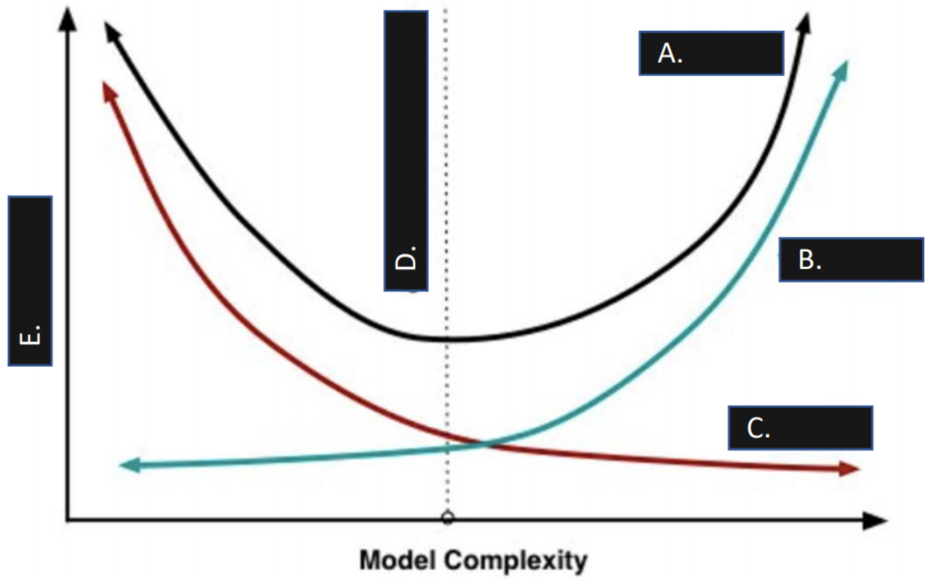
Posted: 9/19/2023 | Due: 10/2/2023 by 11:59 pm

Instructions: Please submit both the .ipynb file and a PDF version or html. Word documents are not acceptable.

In [1]:
```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from plotnine import *
from statsmodels.formula.api import ols
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

In [2]:
```
def abline(intercept, slope):
    """Plot a line from slope and intercept"""
    axes = plt.gca()
    x_vals = np.array(axes.get_xlim())
    y_vals = intercept + slope * x_vals
    plt.plot(x_vals, y_vals, '--')
```

Question 1 (9 points)



F. G. H. I.

$$\mathrm{E}\left[\left(y-\hat{f}(x)\right)^2\right] = \mathrm{Bias}\left[\hat{f}(x)\right]^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$$

Each of the elements is defined below

A) Total error
B) Variance
C) Bias
D) Optimal model complexity
E) Error
F) Mean Squared Error (MSE)
G) Bias Squared
H) Variance
I) Irreducible Error

Please answer the following questions

1(a) (4 points)

Explain what is the difference between the components (F), (H), and (I)

Answer: MSE measures the average squared difference between predicted and actual values, providing an overall view of a model's predictive accuracy. Variance indicates how much predictions would change if using a different training dataset from the same population; high variance suggests a model might be overfitted to the training data.

Irreducible error represents the unavoidable noise present in any dataset due to factors like unmeasured variables or inherent randomness.

1(b) (4 points)

Explain how does a flexible model differ from a restrictive (i.e., less flexible) model with respect to the elements shown in the figure above.

Answer: Flexible models, with their ability to fit complex data patterns, typically have low bias but risk high variance by potentially overfitting to training data noise. Restrictive models, being simpler, might have higher bias due to missing underlying data intricacies but benefit from lower variance, thus yielding more consistent predictions across various datasets. Both types of models are subject to irreducible error, an inherent dataset noise. Total error encompasses bias, variance, and irreducible error. The goal is to find an optimal model complexity that minimizes the total error, balancing the trade-offs between bias and variance without excessively adapting to the data or being overly simplistic.

1(c) (1 point)

Give one example of a more flexible model and one example of a less flexible model.

Answer: KNN, especially with a smaller k, is a flexible model that can adapt to intricate data patterns but risks overfitting. In contrast, linear regression assumes a linear relationship, making it inherently less flexible and potentially unable to capture complex, non-linear trends, though it offers stability and interpretability.

Question 2 (10 points)

Keeping the above picture in Q1 in mind, answer the following questions:

2(a) (2 points)

Explain in a few sentences what does expectation $E[(y - \hat{f}(x))^2]$ over the sample data really mean?

Answer: The expression represents the expected value of the squared difference between the actual value y and the predicted value $\hat{f}(x)$ over the sample data. This is commonly referred to as the Mean Squared Error (MSE). In essence, the MSE quantifies the average squared discrepancy between predicted and true values, giving a measure of the model's prediction accuracy.

2(b) (2 points)

Why would you consider a less flexible model over a more flexible one? Give specific reasons (explain in writing and/or give an example to illustrate it)

Answer: Choosing a less flexible model over a more flexible one can be beneficial for several reasons. Less flexible models are often easier to understand and interpret, making them preferable when clarity is essential. They are also less prone to overfitting, meaning they won't excessively adapt to noise in the training data, ensuring better generalization to new data. Additionally, they are computationally more efficient and can provide more stable predictions. For instance, when predicting house prices based on the number of bedrooms, if the trend is largely linear, a simple linear regression might suffice. Using a complex model, like a neural network, might overfit to minor fluctuations in the training data, leading to inconsistent predictions for new houses.

2(c) (2 point)

Does the bias-variance trade-off apply to unsupervised learning problems? Justify your answer.

Answer: The bias-variance tradeoff does apply to unsupervised learning problems. This is because unsupervised learning algorithms still make assumptions and generalizations about the underlying structure of the data, which can lead to underfitting or overfitting.

For example, k-means clustering makes the assumption that the data can be partitioned into a predefined number of clusters with spherical Gaussian distributions. If the number of clusters is set too low, the model will fail to capture real subgroups in the data, leading to high bias. If k is set too high, the model will overfit by partitioning natural clusters into smaller sub-clusters, leading to high variance. The optimal k must be selected to balance bias and variance.

Principal component analysis (PCA) also faces this tradeoff. Using too few principal components can lead to high bias, failing to capture enough of the dataset variation. Using too many components can lead to overfitting, modeling noise rather than the true latent factors. The number of components must be tuned, often using cross-validation, to optimize this tradeoff.

In summary, unsupervised learning algorithms make inductive assumptions and trade off the flexibility to closely fit the data with the risk of modeling noise. This leads to the universal bias-variance tradeoff, even for unsupervised problems without ground truth labels. Careful model selection and cross-validation can help balance bias and variance.

2(d) (2 points)

What do we expect will happen to bias and variance when we increase the sample size and the model is less flexible?

Answer: When increasing the sample size in a dataset, the bias of a less flexible model is expected to remain relatively stable, since bias is primarily a reflection of the model's inherent assumptions and its ability to capture underlying patterns in the data, irrespective of sample size. On the other hand, variance, which measures the model's sensitivity to fluctuations in the training data, is anticipated to decrease with a larger sample size. This is because a greater amount of data tends to provide a more comprehensive representation of the underlying distribution, thereby reducing the likelihood that the model will overfit to random noise or specificities in a smaller subset of the data. Consequently, increasing the sample size will generally bolster the robustness of a less flexible model by mitigating its variance without significantly affecting its bias.

2(e) (2 points)

If you are fitting a model where the output has two categories (also called levels), how would you evaluate the performance of your model? Explain.

Answer: When fitting a model where the output has two categories or levels, several metrics are essential for evaluating its performance. The error rate, or the proportion misclassified, quantifies the percentage of predictions that the model gets incorrect, providing a direct insight into the model's accuracy. Sensitivity, often referred to as the true positive rate, measures the proportion of actual positive cases that the model correctly identifies, indicating the model's ability to detect the presence of a particular category. Specificity, on the other hand, represents the true negative rate and assesses the proportion of actual negative cases that the model correctly classifies. Together, sensitivity and specificity offer a comprehensive view of the model's discriminative ability for both categories, ensuring that the performance evaluation is balanced and takes into consideration both false positives and false negatives.

Question 3 (22 points)

In simple linear regression, we model our prediction $\hat{y}$ as a function of x.

$$\hat{y}_i = \beta_1 x_i + \beta_0 + \epsilon_i$$

3(a) (2 points)

What are the maximum likelihood estimates for β_1 and β_0?

Answer: Least Squares Estimates (LSE) obtained by minimizing Mean Square Error (MSE) for β_1 and β_0 are: $\hat{\beta}_1 = \Sigma(x_i - \bar{x})(y_i - \bar{y})/\Sigma(x_i - \bar{x})^2$; and $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$

3(b) (4 points)

Create functions to calculate β_1 and β_0 given an x vector and a y vector using the MLE equations you found above.

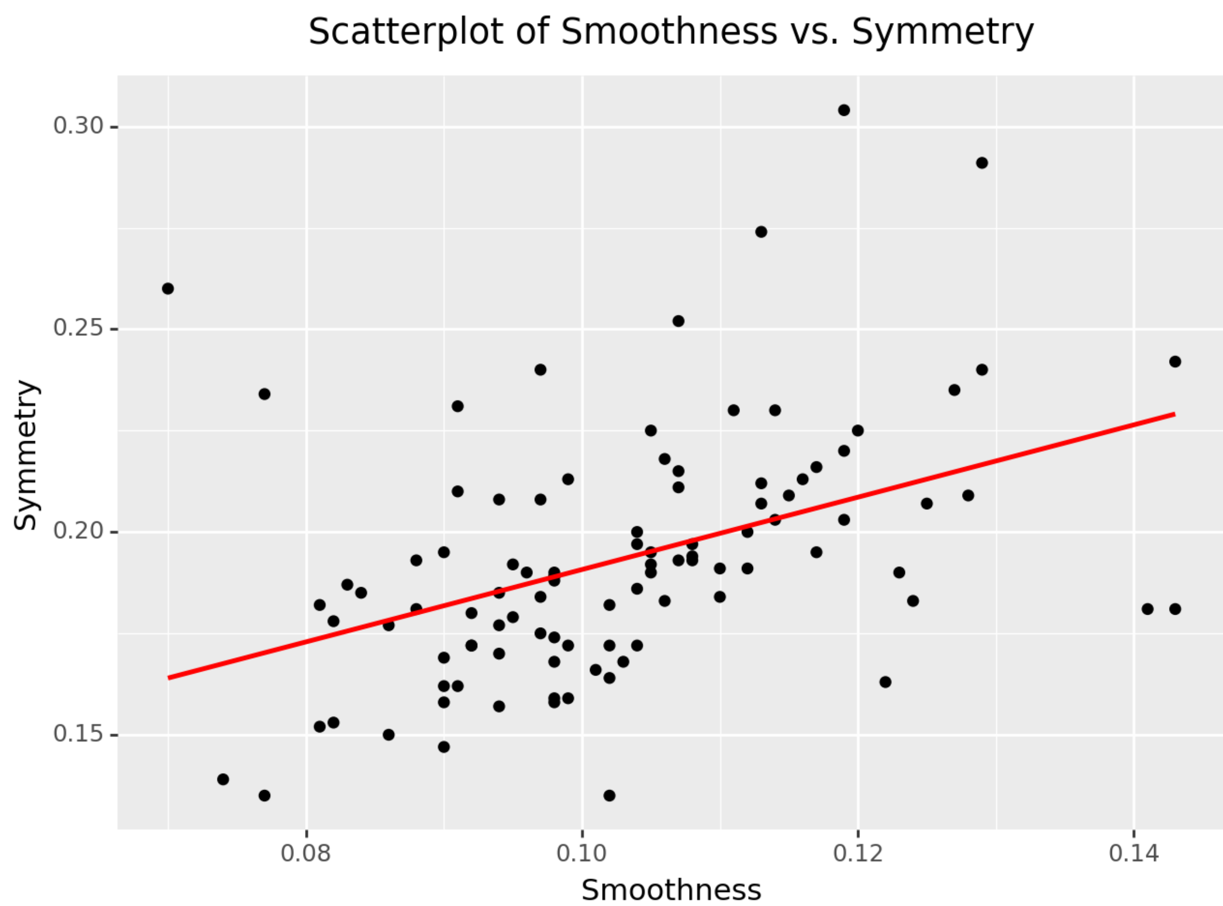
In [3]:
```
# Please remove the "pass" statement from the function and proceed with your ow
# Write your answer below
def getEstimates(x, y):
    x_bar = np.mean(x)
    y_bar = np.mean(y)

    b1 = np.sum((x - x_bar) * (y - y_bar)) / np.sum((x - x_bar)**2)
    b0 = y_bar - b1 * x_bar

    return b0, b1

# Test:
b0 = 1
b1 = 2
x = np.arange(1, 11)
y = b0 + b1*x

getEstimates(x = x, y = y)
```

Out[3]:
```
(1.0, 2.0)
```

3(c) (2+2 points)

Load the `prostate_cancer.csv` data set from Canvas. Use ggplot from plotnine library to draw a scatterplot of smoothness x against symmetry y. Also draw a linear model through the points. Label your figure appropriately. Interpret what you find from the scatterplot and the straight line drawn through them.

In [4]:
```
# Write your answer below
# Load the dataset
data = pd.read_csv('prostate_cancer.csv')

# Scatterplot with linear regression
plot = (
    ggplot(data, aes(x='smoothness', y='symmetry')) +
    geom_point() +
    geom_smooth(method='lm', color='red', se=False) +
    labs(title="Scatterplot of Smoothness vs. Symmetry",
         x="Smoothness",
         y="Symmetry") +
    theme(plot_title=element_text(hjust=0.5)) # Center the title
)

print(plot)
```



Interpretation: The scatterplot of smoothness against symmetry showcases a general upward trend, indicating a positive correlation between the two variables: as smoothness increases, symmetry tends to also increase. This observation is emphasized by the upward-sloping linear regression line. However, the points are relatively sparse with some notable outliers, suggesting variability in the data and cautioning against broad generalizations based solely on the trend line.

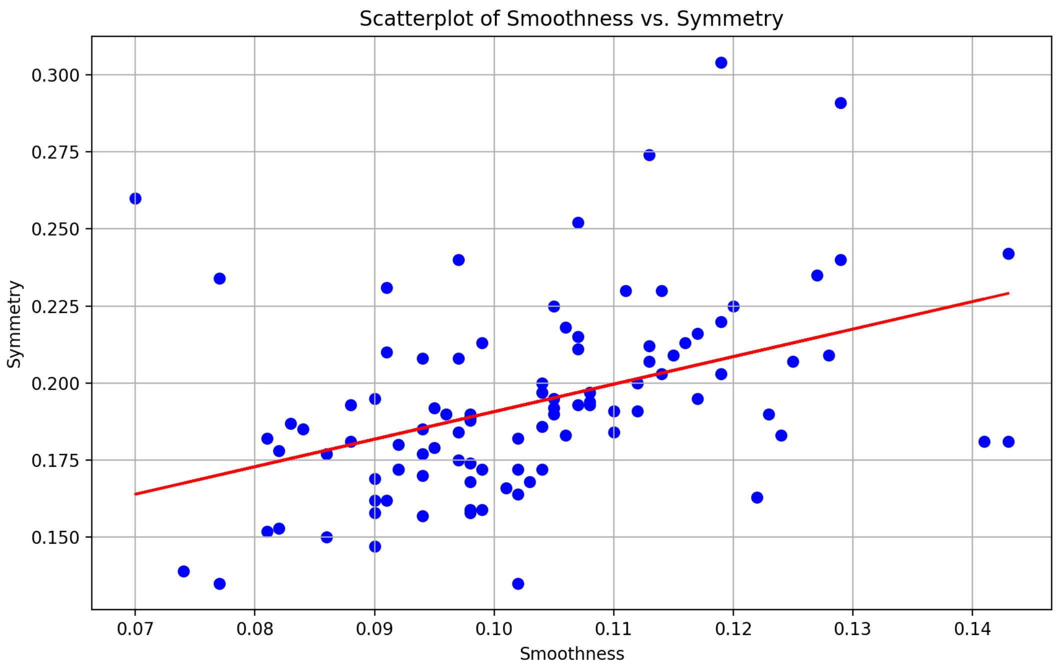
3(d) (3 points)

Use the maximum-likelihood estimates of β_0 and β_1 given the data (smoothness x and symmetry y). Use matplotlib to plot the data and the best-fit line using these estimates (hint: use geom_abline()). As always, label your figure appropriately.

Compare your results obtained in 3(c) and 3(d). They should be the same.

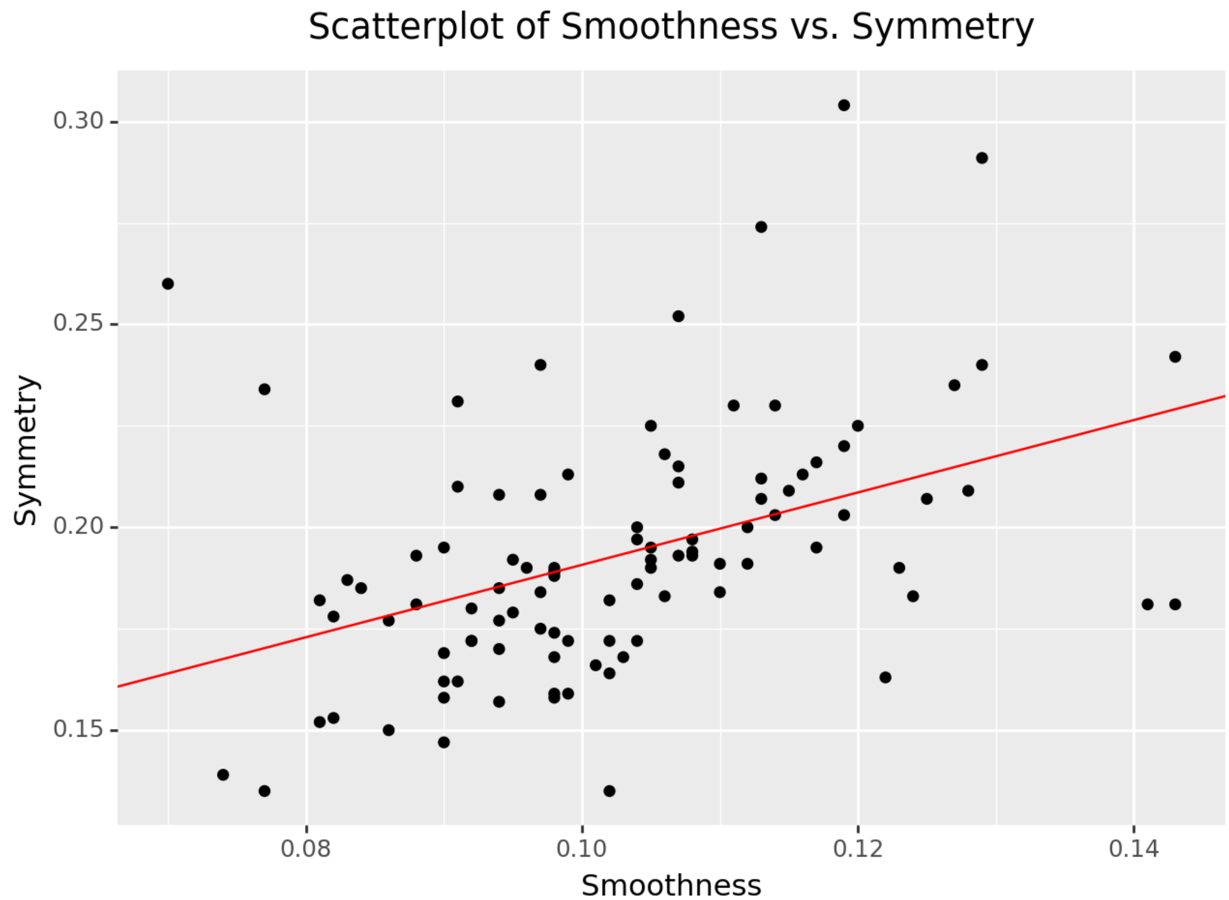
In [5]:
```
# Write your answer below
# Get the intercept (b0) and slope (b1) for the prostate cancer data
b0, b1 = getEstimates(data['smoothness'], data['symmetry'])

# Plot the data using matplotlib
plt.figure(figsize=(10, 6))
plt.scatter(data['smoothness'], data['symmetry'], color='blue')
plt.plot(data['smoothness'], b0 + b1 * data['smoothness'], color='red')
```

```
plt.title("Scatterplot of Smoothness vs. Symmetry")
plt.xlabel("Smoothness")
plt.ylabel("Symmetry")
plt.grid(True)
plt.show()
```



In [6]:

```
# Plot the data using ggplot (geom_abline())
plot = (
    ggplot(data, aes(x='smoothness', y='symmetry')) +
    geom_point() +
    geom_abline(intercept=b0, slope=b1, color='red') +
    labs(title="Scatterplot of Smoothness vs. Symmetry",
         x='Smoothness',
         y='Symmetry') +
    theme(plot_title=element_text(hjust=0.5)) # Center the title
)

print(plot)
```



3(e) (4 points)

Give a 95% confidence interval for β_0 and β_1. The formula for confidence intervals for $\hat{\beta}_0$ and $\hat{\beta}_1$ involves calculation of standard errors.

Once you obtain those, write a Python function conf_betas() to obtain the confidence intervals.

Answer

In [7]:

```
# Write your answer below
from scipy.stats import t

def conf_betas(x, y, alpha=0.05):
    n = len(x)

    # Get the estimates for b0 and b1
    b0, b1 = getEstimates(x, y)

    # Calculate residuals and RSS
    e = y - (b0 + b1 * x)
    rss = np.sum(e**2)

    # Compute standard errors
    se_b1 = np.sqrt(rss / ((n-2) * np.sum((x - np.mean(x))**2)))
    se_b0 = se_b1 * np.sqrt(1/n + np.mean(x)**2 / np.sum((x - np.mean(x))**2))
```

```python
    # Calculate confidence intervals using t-distribution
    t_stat = t.ppf(1-alpha/2, n-2)
    ci_b1 = [b1 - t_stat * se_b1, b1 + t_stat * se_b1]
    ci_b0 = [b0 - t_stat * se_b0, b0 + t_stat * se_b0]

    return ci_b0, ci_b1

# Test:
x = data['smoothness']
y = data['symmetry']

ci_b0, ci_b1 = conf_betas(x, y)
print(f"95% Confidence Interval for b0: {ci_b0}")
print(f"95% Confidence Interval for b1: {ci_b1}")
```

```
95% Confidence Interval for b0: [-0.17029265704868496, 0.37338130460370716]
95% Confidence Interval for b1: [0.5102286924114292, 1.2735866725742404]
```

3(f) (4 points)

Use the statsmodels `ols()` function to fit a linear model `lmFit`. Print the summary statistics for the model. Plot the data and use the `abline()` function to plot the best fit line.

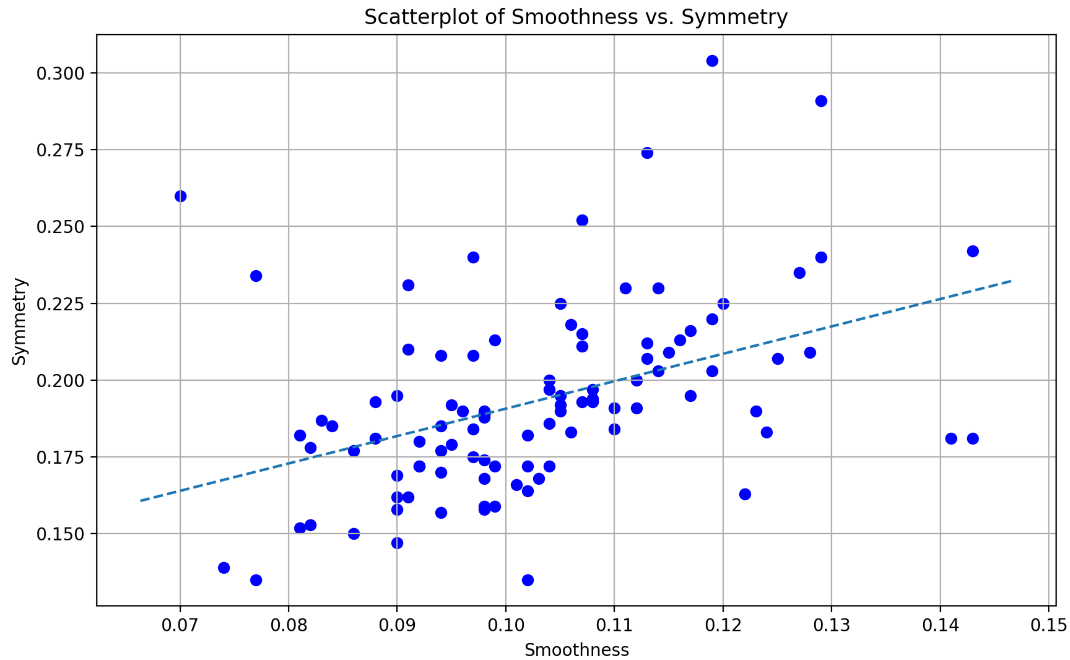
In [8]:

```python
# Write your answer below
# Fit the linear model
lmFit = ols('symmetry ~ smoothness', data=data).fit()

# Print the summary statistics
print(lmFit.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:               symmetry   R-squared:                       0.180
Model:                            OLS   Adj. R-squared:                  0.172
Method:                 Least Squares   F-statistic:                     21.50
Date:                Sun, 01 Oct 2023   Prob (F-statistic):           1.09e-05
Time:                        02:16:23   Log-Likelihood:                 216.60
No. Observations:                 100   AIC:                            -429.2
Df Residuals:                      98   BIC:                            -424.0
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.1015      0.020      5.088      0.000       0.062       0.141
smoothness     0.8919      0.192      4.637      0.000       0.510       1.274
==============================================================================
Omnibus:                       26.568   Durbin-Watson:                   2.037
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               42.850
Skew:                           1.157   Prob(JB):                     4.96e-10
Kurtosis:                       5.220   Cond. No.                         69.4
==============================================================================

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

In [9]:

```
# Plot here
plt.figure(figsize=(10, 6))
plt.scatter(data['smoothness'], data['symmetry'], color='blue')
abline(lmFit.params['Intercept'], lmFit.params['smoothness'])

plt.title('Scatterplot of Smoothness vs. Symmetry')
plt.xlabel('Smoothness')
plt.ylabel('Symmetry')
plt.grid(True)
plt.show()
```



3(g) (1 point)

Print the 95% prediction interval of the symmetry for a tumor with smoothness $x = 0.13$. (Hint: Use conf_int function)

In [10]:
```
x0 = 0.13

# Get the prediction and confidence interval
pred = lmFit.get_prediction({'smoothness': x0})
pred_int = pred.conf_int(alpha=0.05)  # 95% confidence level

# Extract the lower and upper bounds of the prediction interval
lower_bound = pred_int[0, 0]
upper_bound = pred_int[0, 1]

print(f"95% Prediction Interval: [{lower_bound}, {upper_bound}]")
```

```
95% Prediction Interval: [0.20569177347489254, 0.22929287152826638]
```

Question 4 (18 points)

We will explore multiple linear regression using the Real Estate Valuation data set uploaded to HW2 folder on Canvas.

4(a) (3 points)

The market historical data set of real estate valuation are collected from Sindian Dist., New Taipei City, Taiwan

The inputs are as follows

transaction_date=the transaction date (for example, 2013.250=2013 March, 2013.500=2013 June, etc.)
house_age = the house age (unit: year)
distance_station=the distance to the nearest MRT station (unit: meter)
num_conv_stores=the number of convenience stores in the living circle on foot (integer)
latitude=the geographic coordinate, latitude. (unit: degree)
longitude=the geographic coordinate, longitude. (unit: degree)

The output is as follows

price_per_unit_area= house price of unit area (10000 New Taiwan Dollar/Ping, where Ping is a local unit, 1 Ping = 3.3 meter squared)

Our goal is to fit a multiple linear regression model to predict the price per unit of area of the houses. Note that there are some variables in the data set that are not meaningful without further processing for modeling the house price. You should exclude them.*italicised text*

In [11]:
```python
# Read the XLSX file.You have use the pd.read_excel function (1)
df = pd.read_excel('RealEstateValuation.xlsx')

# Show the dataframe (1)
print(df)
```

```
     transaction_date  house_age  distance_station  num_conv_stores  latitude  \
0         2012.916667       32.0          84.87882               10  24.98298
1         2012.916667       19.5         306.59470                9  24.98034
2         2013.583333       13.3         561.98450                5  24.98746
3         2013.500000       13.3         561.98450                5  24.98746
4         2012.833333        5.0         390.56840                5  24.97937
..                ...        ...               ...              ...       ...
409       2013.000000       13.7        4082.01500                0  24.94155
410       2012.666667        5.6          90.45606                9  24.97433
411       2013.250000       18.8         390.96960                7  24.97923
412       2013.000000        8.1         104.81010                5  24.96674
413       2013.500000        6.5          90.45606                9  24.97433

     longitude  price_per_unit_area
0    121.54024                 37.9
1    121.53951                 42.2
2    121.54391                 47.3
3    121.54391                 54.8
4    121.54245                 43.1
..         ...                  ...
409  121.50381                 15.4
410  121.54310                 50.0
411  121.53986                 40.6
412  121.54067                 52.5
413  121.54310                 63.9

[414 rows x 7 columns]
```

In [12]:
```python
# Show head of the data set (1 point)
df.head()
```

Out[12]:

	transaction_date	house_age	distance_station	num_conv_stores	latitude	longitude	price
0	2012.916667	32.0	84.87882	10	24.98298	121.54024	
1	2012.916667	19.5	306.59470	9	24.98034	121.53951	
2	2013.583333	13.3	561.98450	5	24.98746	121.54391	
3	2013.500000	13.3	561.98450	5	24.98746	121.54391	
4	2012.833333	5.0	390.56840	5	24.97937	121.54245	

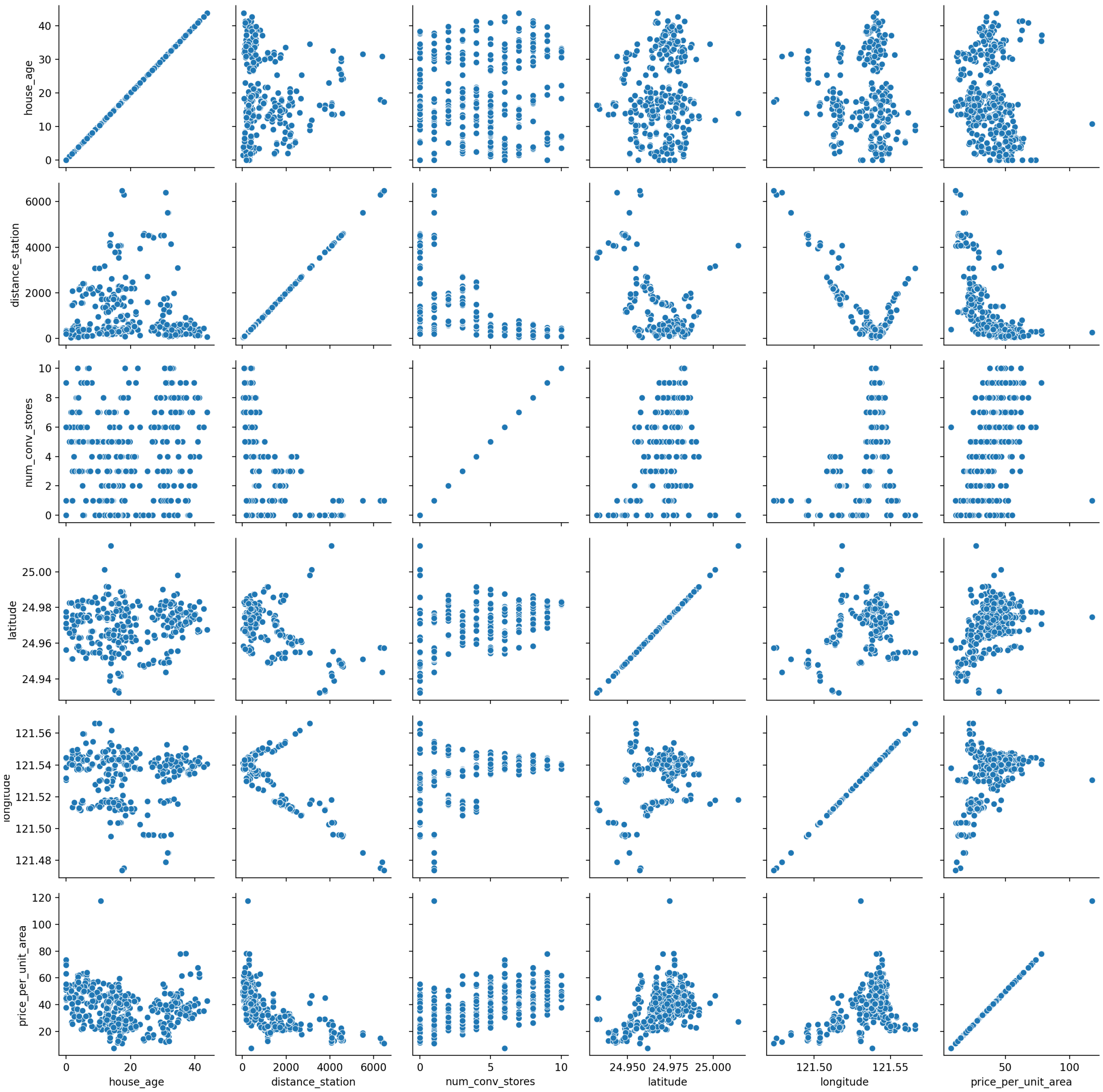
4(b) (2 points)

Create a subset of the data set by using only the relevant variables for modeling purposes. Produce a scatterplot matrix. Which variables are most correlated (positively and negatively) with the house price?

In [13]:
```python
# Create working data set (1 point)
df_ = df.drop(columns=['transaction_date'])
```

In [14]:
```python
# Pairs plot with sns.PairGrid or any suitable package (1 point)
g = sns.PairGrid(df_)
g.map(sns.scatterplot)
```

Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x78f9ec0aef50>
```



In [15]:
```python
# Produce the correlation matrix
correlation_matrix = df_.corr()
print(correlation_matrix)
```

```
                     house_age  distance_station  num_conv_stores  latitude  \
house_age             1.000000          0.025622         0.049593  0.054420
distance_station      0.025622          1.000000        -0.602519 -0.591067
num_conv_stores       0.049593         -0.602519         1.000000  0.444143
latitude              0.054420         -0.591067         0.444143  1.000000
longitude            -0.048520         -0.806317         0.449099  0.412924
price_per_unit_area  -0.210567         -0.673613         0.571005  0.546307

                     longitude  price_per_unit_area
house_age            -0.048520            -0.210567
distance_station     -0.806317            -0.673613
num_conv_stores       0.449099             0.571005
latitude              0.412924             0.546307
longitude             1.000000             0.523287
price_per_unit_area   0.523287             1.000000
```

In [16]:
```
print(correlation_matrix["price_per_unit_area"].sort_values(ascending=False))
```

```
price_per_unit_area    1.000000
num_conv_stores        0.571005
latitude               0.546307
longitude              0.523287
house_age             -0.210567
distance_station      -0.673613
Name: price_per_unit_area, dtype: float64
```

Answer: The variable most positively correlated with the house price is `num_conv_stores` with a correlation of 0.571005.

The variable most negatively correlated with the house price is `distance_station` with a correlation of -0.673613.

This implies that as the number of convenience stores increases, the price per unit area tends to increase, and as the distance to the nearest MRT station increases, the price per unit area tends to decrease.

4(c) (2 points)

Fit a multiple linear regression model to predict price_per_unit_area using the meaningful variables and produce a summary.

In [17]:
```
# Write your answer below
# Fit the model
model = ols('price_per_unit_area ~ num_conv_stores + latitude + longitude + hou

# View the summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:     price_per_unit_area   R-squared:                       0.57
1
Model:                             OLS   Adj. R-squared:                  0.56
6
Method:                  Least Squares   F-statistic:                     108.
7
Date:                 Sun, 01 Oct 2023   Prob (F-statistic):           9.34e-7
3
Time:                         02:16:34   Log-Likelihood:                -1492.
4
No. Observations:                  414   AIC:                             299
7.
Df Residuals:                      408   BIC:                             302
1.
Df Model:                            5
Covariance Type:             nonrobust
==============================================================================
======
                      coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
------
Intercept       -4945.5951   6211.157     -0.796      0.426   -1.72e+04     72
64.269
num_conv_stores     1.1630      0.190      6.114      0.000       0.789      
1.537
latitude          237.7672     44.948      5.290      0.000     149.409      3
26.126
longitude          -7.8055     49.149     -0.159      0.874    -104.422      
88.811
house_age          -0.2689      0.039     -6.896      0.000      -0.346      
-0.192
distance_station   -0.0043      0.001     -5.888      0.000      -0.006      
-0.003
==============================================================================
Omnibus:                      240.068   Durbin-Watson:                   2.149
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3748.747
Skew:                           2.129   Prob(JB):                         0.00
Kurtosis:                      17.114   Cond. No.                     2.35e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 2.35e+07. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

4(d) (3 points)

Using the summary statistics F, r^2, p, explain whether or not there is a relationship between the predictors and the response variable. Which, if any, of the predictors are significant?

In [18]:
```
# Write your answer below
# Give Fvalue
```

```python
print(model.fvalue)
```

```
108.68151457914628
```

In [19]:
```python
# Give rsquared (1)
print(model.rsquared)
```

```
0.5711617064827438
```

In [20]:
```python
# Give rsquared adjusted (1)
print(model.rsquared_adj)
```

```
0.5659063352386597
```

In [21]:
```python
# Give p value of f (1)
print(model.f_pvalue)
```

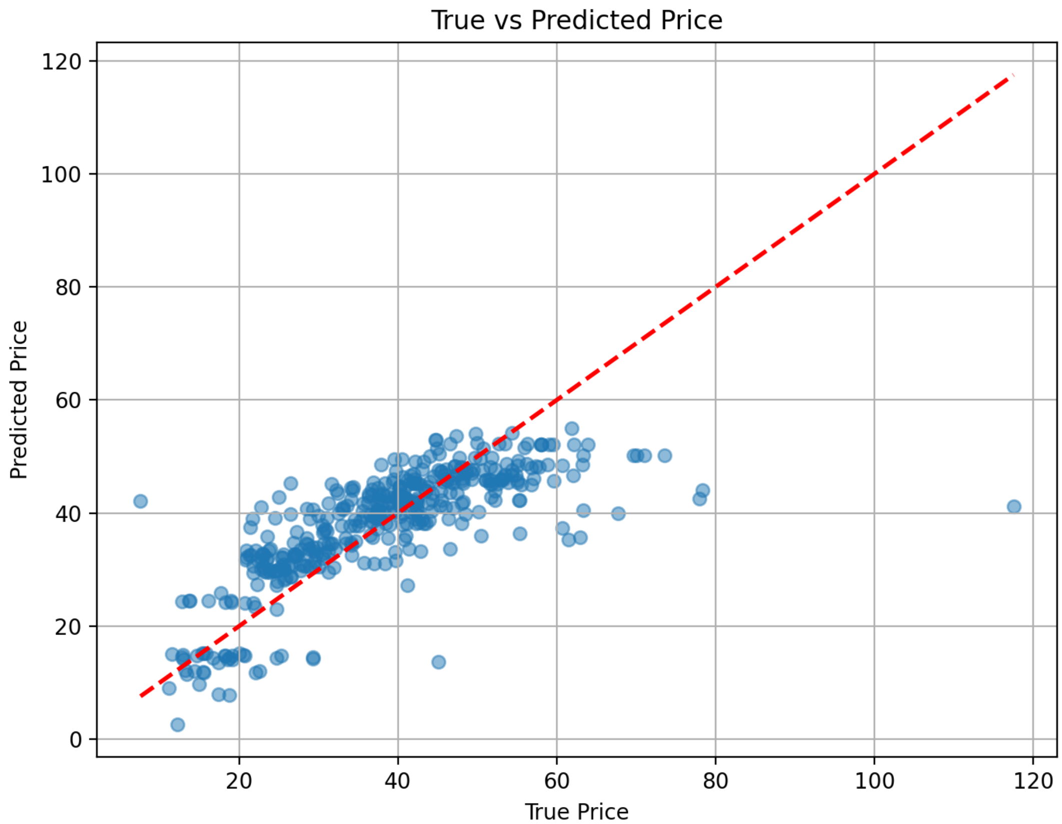
```
9.336758847465961e-73
```

Answer: The model has an F-value of 108.7 and a near-zero p-value (9.34e-73), indicating a significant relationship between the predictors and the response variable. Furthermore, the model explains 57.1% of the variance in house prices, as evidenced by the R-squared value. The adjusted R-squared, which is closely behind at 0.566, shows that the model's predictors are indeed relevant. When examining individual predictors, `num_conv_stores`, `latitude`, `house_age`, and `distance_station` stand out as significant contributors to the model, each with p-values close to zero. On the other hand, `longitude` does not appear to have a significant impact on the house price given its high p-value of 0.874. In essence, most of the selected predictors significantly influence house prices in the dataset, except for `longitude`.

4(e) (3 points)

Plot the true price_per_unit_area against the estimated price_per_unit_area. Plot the diagonal line. Where does the model seem to underestimate price_per_unit_area? Where does the model seem to overestimate price?

In [22]:
```python
# Write your answer below
# Predicting the prices using the model
predicted_prices = model.predict(df_)

# Plotting the true prices against predicted prices
plt.figure(figsize=(8, 6))
plt.scatter(df_['price_per_unit_area'], predicted_prices, alpha=0.5)
plt.plot([min(df_['price_per_unit_area']), max(df_['price_per_unit_area'])],
         [min(df_['price_per_unit_area']), max(df_['price_per_unit_area'])],
         'r--', lw=2)
plt.xlabel('True Price')
plt.ylabel('Predicted Price')
plt.title('True vs Predicted Price')
plt.grid(True)
plt.show()
```



In [23]:
```
# Calculate number of overestimated instances
overestimated = np.sum(predicted_prices > df_['price_per_unit_area'])

# Calculate number of underestimated instances
underestimated = np.sum(predicted_prices < df_['price_per_unit_area'])

# Calculate percentages
overestimated_percentage = (overestimated / len(df_)) * 100
underestimated_percentage = (underestimated / len(df_)) * 100

print(f"Percentage of overestimation: {overestimated_percentage:.2f}%")
print(f"Percentage of underestimation: {underestimated_percentage:.2f}%")
```

```
Percentage of overestimation: 57.49%
Percentage of underestimation: 42.51%
```

Answer: The model tends to overestimate house prices, with an overestimation percentage of 57.49%, particularly when the actual price per unit area is below 50. However, for houses priced higher than 50 per unit area, the model leans towards underestimation, accounting for the remaining 42.51%. In essence, while the model generally overestimates more frequently, lower-priced houses are more prone to overestimation, whereas higher-priced houses are more likely to be underestimated.

4(f) (2 points)

Should we have used a more flexible model? If so, what kind?

Answer: The model's tendency to overestimate lower house prices and underestimate higher ones suggests potential non-linearities in the relationship between predictors and the response, which the current linear model might not adequately capture.

A more flexible model might be beneficial. One potential approach is polynomial regression, which introduces higher-order terms to capture curvilinear relationships. However, this method carries a risk of overfitting, especially if too many polynomial terms are added without discretion.

Alternatively, tree-based models like decision trees, random forests, or gradient boosting machines can be explored. These models can capture non-linear relationships and interactions between predictors. They work by segmenting the data into regions, offering a solution to the observed overestimation and underestimation problem.

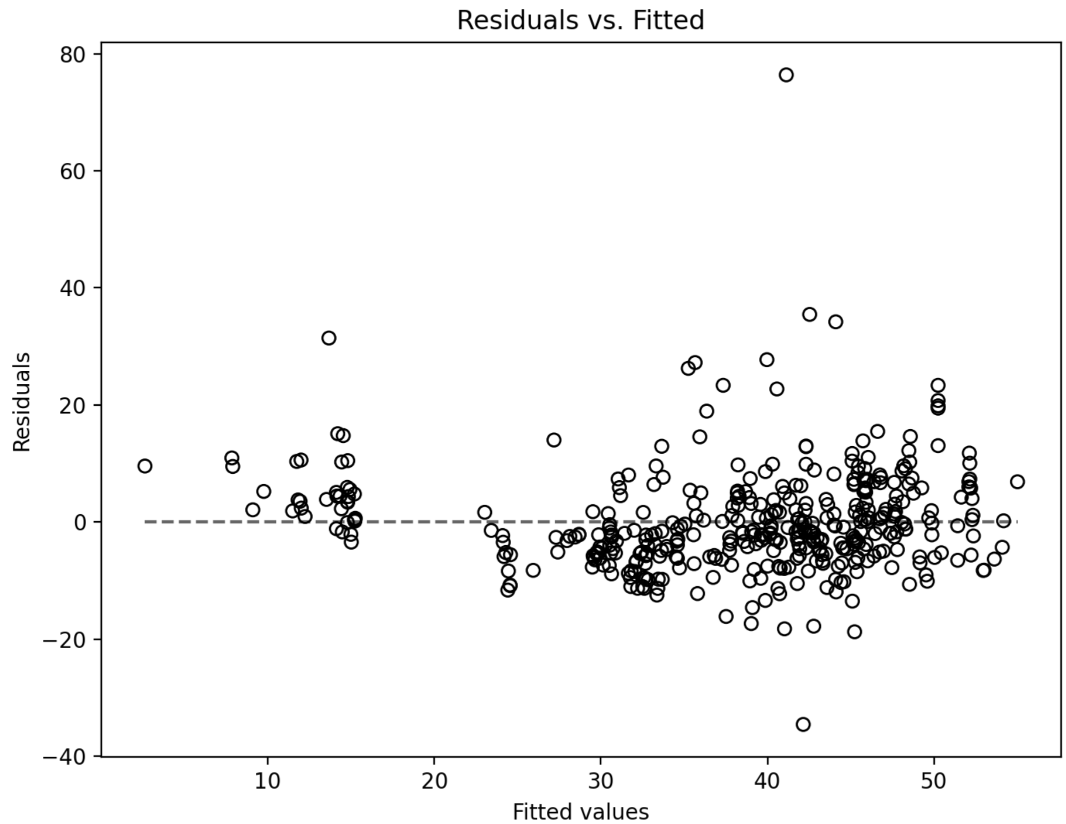
In conclusion, while the current model provides some insight, there's room for refinement. Exploring more flexible modeling techniques, employing cross-validation to prevent overfitting, and incorporating domain knowledge can lead to a more accurate representation of the housing market's complexities.

4(g) (3 points)

Produce diagnostic plots of your model. Can you detect any outliers? Justify eliminating those outliers.

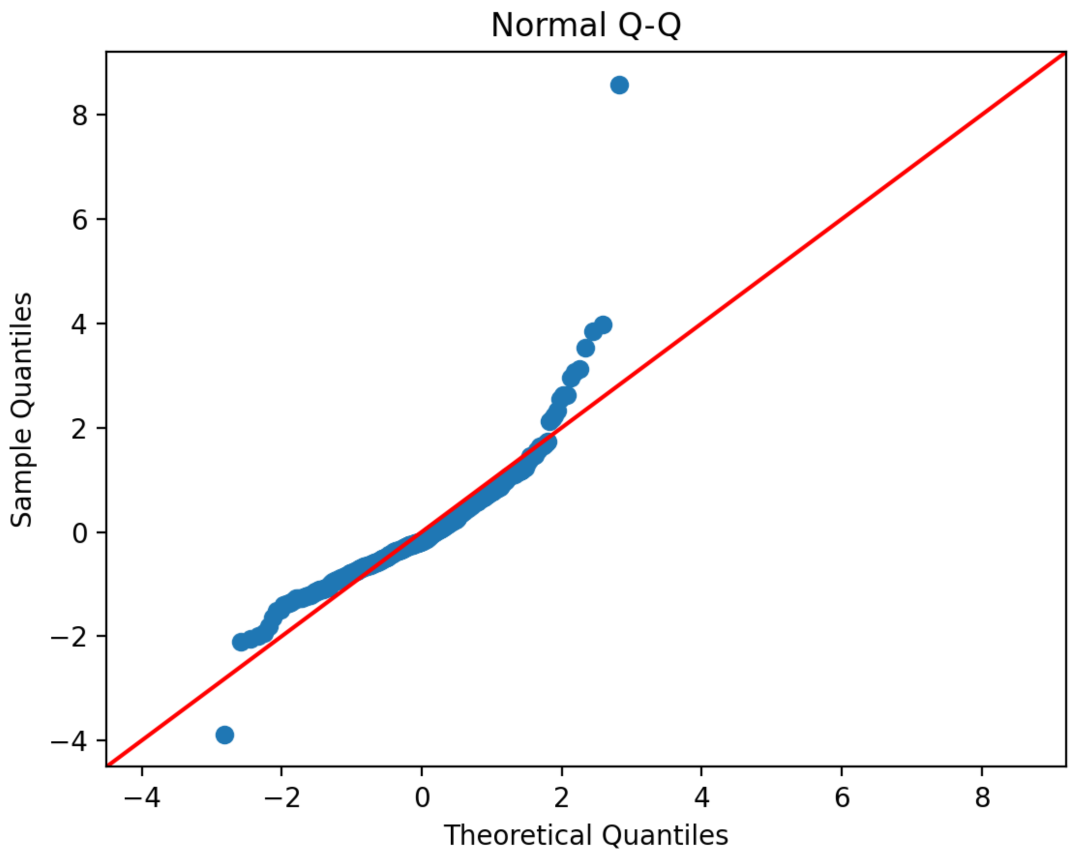
In [24]:
```
# Write your answer below
import statsmodels.api as sm

# Get the residuals and fitted values
residuals = model.resid
fitted = model.fittedvalues
normalized_residuals = model.get_influence().resid_studentized_internal
leverage = model.get_influence().hat_matrix_diag
cooks_distance = model.get_influence().cooks_distance[0]
```

In [25]:
```
# Residuals vs Fitted
plt.figure(figsize=(8, 6))
plt.scatter(fitted, residuals, edgecolors='k', facecolors='none')
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted')
plt.plot([min(fitted), max(fitted)], [0, 0], color='k', linestyle='--', alpha=(
plt.show()
```

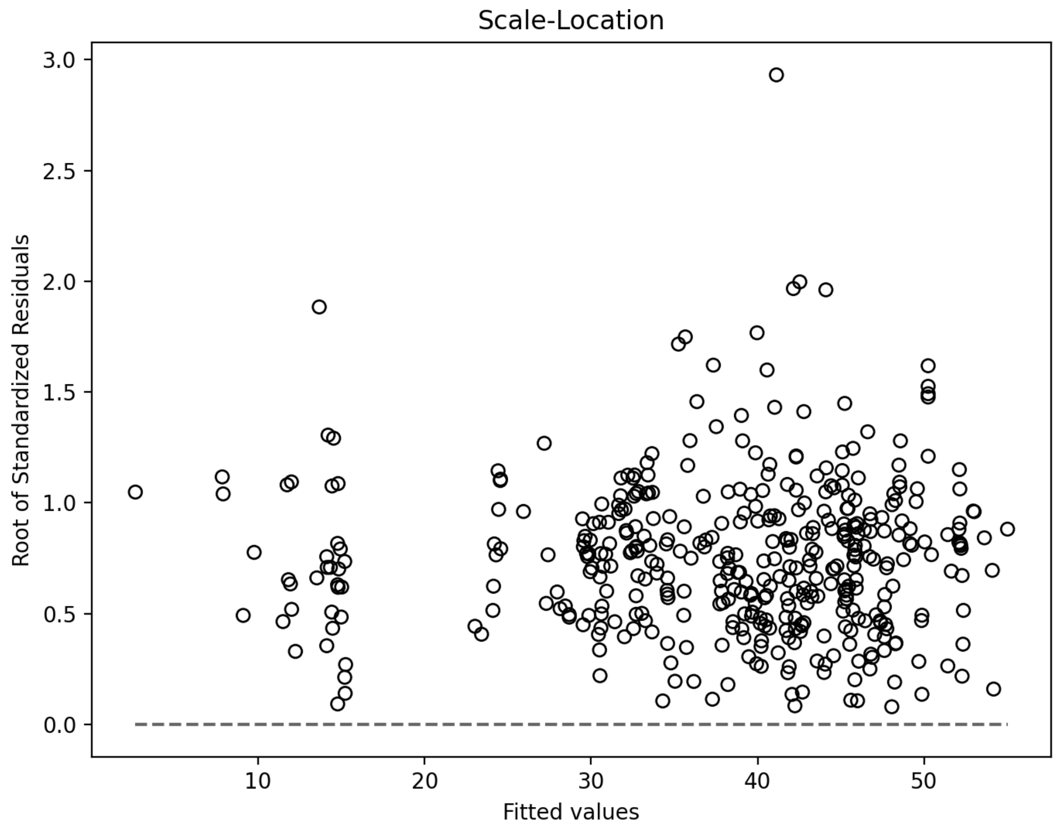


In [26]:
```
# Normal Q-Q
plt.figure(figsize=(8, 6))
sm.qqplot(residuals, line='45', fit=True)
plt.title('Normal Q-Q')
plt.show()
```

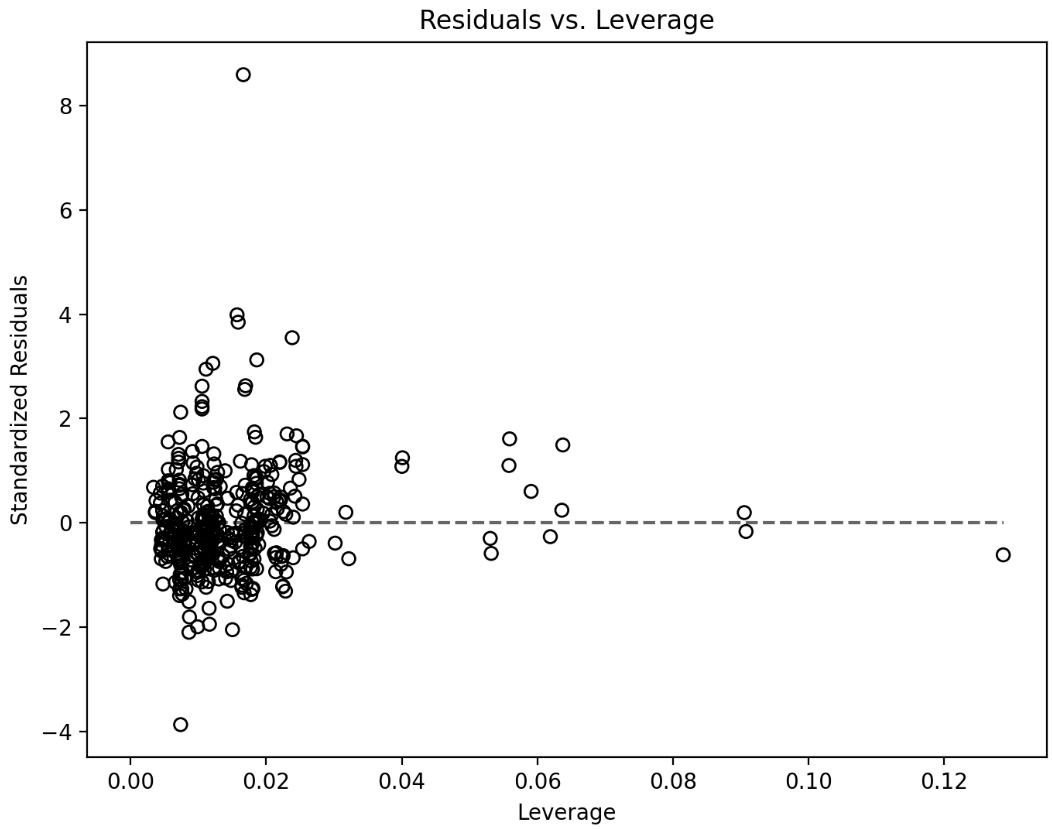
```
<Figure size 800x600 with 0 Axes>
```



In [27]:
```
# Scale-Location
plt.figure(figsize=(8, 6))
plt.scatter(fitted, abs(normalized_residuals)**0.5, edgecolors='k', facecolors=
plt.xlabel('Fitted values')
plt.ylabel('Root of Standardized Residuals')
plt.title('Scale-Location')
plt.plot([min(fitted), max(fitted)], [0, 0], color='k', linestyle='--', alpha=(
plt.show()
```



In [28]:
```
# Residuals vs. Leverage
plt.figure(figsize=(8, 6))
plt.scatter(leverage, normalized_residuals, edgecolors='k', facecolors='none')
plt.xlabel('Leverage')
plt.ylabel('Standardized Residuals')
plt.title('Residuals vs. Leverage')
plt.plot([0, max(leverage)], [0, 0], color='k', linestyle='--', alpha=0.6)
plt.show()
```



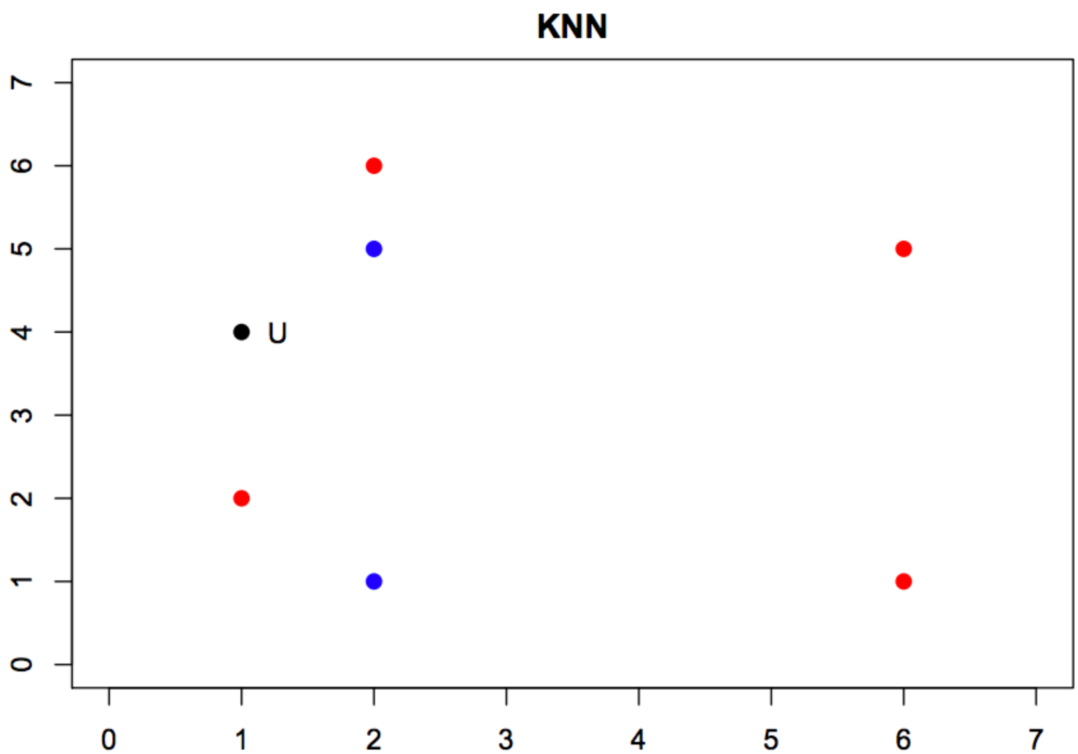
In [29]:
```
# Setting the threshold based on the rule of thumb
threshold = 4 / (len(df_) - 4 - 1) # 4 significant predictors

# Detecting outliers based on Cook's distance
outliers = [i for i, v in enumerate(cooks_distance) if v > threshold]
print(f"Potential Outliers based on Cook's distance: {outliers}")
```

```
Potential Outliers based on Cook's distance: [8, 47, 113, 116, 126, 128, 148,
166, 194, 220, 228, 270, 312, 334, 344, 361, 382, 389]
```

Answer: Outliers detected using diagnostic methods like Cook's distance and visual plots can significantly influence the regression model, leading to biased parameter estimates and compromising its predictive accuracy and generalizability. Such outliers might distort the underlying data structure that the model aims to capture. Furthermore, outliers can violate core assumptions of linear regression, such as linearity, normality, and homoscedasticity. Adherence to these assumptions is vital for the model to provide valid inferences. When points significantly deviate from the expected patterns in diagnostic plots, they can unduly leverage the model, skewing its results. Therefore, eliminating these outliers ensures that the model accurately represents the core trends in the data without being unduly influenced by anomalous observations.

Question 5 (8 points)

We will use the following data to explore kNNs.



This figure uses the following data:

In [30]:
```
x = np.array([1, 1, 2, 2, 2, 6, 6])
y = np.array([4, 2, 6, 5, 1, 5, 1])
color = ['black', 'red', 'red', 'blue', 'blue', 'red', 'red']
```

5(a) (2 points)

Compute the Euclidean distance for each point with respect to the unlabeled point, U.

In [31]:
```
# Write your answer below
# Calculate the Euclidean distances using vectorized operations
distances = np.sqrt((x - 1)**2 + (y - 4)**2)

# Print the distances
print(distances)
```

```
[0.         2.         2.23606798 1.41421356 3.16227766 5.09901951
 5.83095189]
```

5(b) (2 points)

What is the kNN label for U when $k = 1$? When $k = 3$?

In [32]:
```
# Write your answer below
# Given data (excluding U)
X = np.array([[1, 2], [2, 6], [2, 5], [2, 1], [6, 5], [6, 1]])
y = np.array(['red', 'red', 'blue', 'blue', 'red', 'red'])
```

```python
U = np.array([[1, 4]])  # Coordinates of the unlabeled point

# Create a KNeighborsClassifier with k = 1
k1_knn_classifier = KNeighborsClassifier(n_neighbors=1)

# Fit the classifier on the labeled data
k1_knn_classifier.fit(X, y)

# Predict the label for the unlabeled point U
k1_predicted_label = k1_knn_classifier.predict(U)

# Print the predicted label for k = 1
print(f"KNN label for U when k = 1: {k1_predicted_label[0]}")
```

```
KNN label for U when k = 1: blue
```

In [33]:
```python
# Create a KNeighborsClassifier with k = 3
k3_knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Fit the classifier on the labeled data
k3_knn_classifier.fit(X, y)

# Predict the label for the unlabeled point U
k3_predicted_label = k3_knn_classifier.predict(U)

# Print the predicted label for k = 3
print(f"KNN label for U when k = 3: {k3_predicted_label[0]}")
```

```
KNN label for U when k = 3: red
```

5(c) (4 point)

It's also possible to calculate kNN using decision functions different from majority vote. One common scheme is to take weighted votes as a function of distance. The procedure is as follows:

- Get the k nearest neighbors.
- Compute the weight for each red neighbor and sum.
- Compute the weight for each blue neighbor and sum.
- Choose the label with larger score.

There are three common weight functions:

- Inverse Euclidean Distance: $||v-u||^{-1}$
- Inverse Square: $||v-u||^{-2}$
- Gaussian Functional Distance: $e^{-\alpha||v-u||^2}$

Using Python, give the kNN for $k=6$ using weighted voting for each of the following weight functions:

- Inverse Euclidean Distance

In [34]:
```python
# Write your answer below
# Create a KNeighborsClassifier with k = 6 and Inverse Euclidean Distance weig
```

```
knn_inv_euc = KNeighborsClassifier(n_neighbors=6, weights='distance')

# Fit the classifier on the labeled data
knn_inv_euc.fit(X, y)

# Predict the label for the unlabeled point U
predicted_label_inv_euc = knn_inv_euc.predict(U)

# Print the predicted label
print(f"KNN label for U with Inverse Euclidean Distance weighting (k = 6): {pre
```

```
KNN label for U with Inverse Euclidean Distance weighting (k = 6): red
```

- Inverse Square

In [35]:
```
# Write your answer below
# Define the custom weight function for Inverse Square
def custom_weight(dist):
    return 1 / (dist ** 2)

# Create a KNeighborsClassifier with k = 6
knn_classifier_inv_sq = KNeighborsClassifier(n_neighbors=6, weights=custom_weig

# Fit the classifier on the labeled data
knn_classifier_inv_sq.fit(X, y)

# Predict the label for the unlabeled point U
predicted_label_inv_sq = knn_classifier_inv_sq.predict(U)

# Print the predicted label
print(f"KNN label for U with Inverse Square weighting (k = 6): {predicted_label
```

```
KNN label for U with Inverse Square weighting (k = 6): blue
```

- Gaussian Functional Distance with $\alpha = 0.2$

In [36]:
```
# Write your answer below
# Define the custom weight function for Gaussian Functional Distance with α = (
def custom_weight(dist):
    alpha = 0.2
    return np.exp(-alpha * dist ** 2)

# Create a KNeighborsClassifier with k = 6
knn_gauss_0_2 = KNeighborsClassifier(n_neighbors=6, weights=custom_weight)

# Fit the classifier on the labeled data
knn_gauss_0_2.fit(X, y)

# Predict the label for the unlabeled point U
predicted_label_gauss_0_2 = knn_gauss_0_2.predict(U)

# Print the predicted label
print(f"KNN label for U with Gaussian Distance (α = 0.2) weighting (k = 6): {pr
```

```
KNN label for U with Gaussian Distance (α = 0.2) weighting (k = 6): red
```

- Gaussian Functional Distance with $\alpha = 0.4$

In [37]:
```python
# Write your answer below
# Define the custom weight function for Gaussian Functional Distance with α = (
def custom_weight(dist):
    alpha = 0.4
    return np.exp(-alpha * dist ** 2)

# Create a KNeighborsClassifier with k = 6
knn_gauss_0_4 = KNeighborsClassifier(n_neighbors=6, weights=custom_weight)

# Fit the classifier on the labeled data
knn_gauss_0_4.fit(X, y)

# Predict the label for the unlabeled point U
predicted_label_gauss_0_4 = knn_gauss_0_4.predict(U)

# Print the predicted label
print(f"KNN label for U with Gaussian Distance (α = 0.4) weighting (k = 6): {pr
```

```
KNN label for U with Gaussian Distance (α = 0.4) weighting (k = 6): blue
```

Question 6 (14 points)

Load the Prostate Cancer data set (attached with the homework on canvas). We will try to predict the abnormal growth of cells using kNN.

6(a) (1 point)

Normalize the data. The scale used for each of the values of the numeric features may be different, hence normalizing data is a good practice. Use the formula for each numeric column c, `(c - min(c)/max(c)-min(c))`.

In [38]:
```python
# Write your answer below
# Create a copy of the original data to store the normalized data
data_normalized = data.copy()

# Normalize numeric columns in the copied dataframe
for column in data_normalized.columns:
    if column not in ['id', 'diagnosis_result']:
        data_normalized[column] = (data_normalized[column] - data_normalized[co

# Display the first few rows of the normalized data
data_normalized.head()
```

Out[38]:

	id	diagnosis_result	radius	texture	perimeter	area	smoothness	compactness	symn
0	1	M	0.8750	0.0625	0.825000	0.448687	1.000000	0.781759	0.63
1	2	B	0.0000	0.1250	0.675000	0.670644	1.000000	0.133550	0.27
2	3	M	0.7500	1.0000	0.650000	0.597255	0.753425	0.397394	0.42
3	4	M	0.3125	0.3125	0.216667	0.109785	0.000000	0.801303	0.73
4	5	M	0.0000	0.5000	0.691667	0.653341	0.972603	0.309446	0.27

6(b) (2 points)

Divide the data into training and testing sets. Use `train_test_split` from `sklearn` and set a random state of 19. Keep 65% examples in your training set and the rest in your test set.

In [39]:

```
# Write your answer below
X = data_normalized.drop(['id', 'diagnosis_result'], axis=1)
y = data_normalized['diagnosis_result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, rando
```

6(c) (4 points)

Use KNeighborsClassifier from sklearn with k = 10 to classify the data. How many are misclassified on the the testing set?

In [40]:

```
# Write your answer below
from sklearn.metrics import accuracy_score

# Initialize the KNeighborsClassifier with k=10
knn = KNeighborsClassifier(n_neighbors=10)

# Train the classifier
knn.fit(X_train, y_train)

# Predict the labels on the test set
y_pred = knn.predict(X_test)

# Calculate the number of misclassified instances
misclassified = (y_test != y_pred).sum()

print(f"{misclassified} misclassified values")
```

```
8 misclassified values
```

6(d) (2 points)

Do this for k = 1,2,3,4,5,6,7,8,9,10,20,30,40,50. (I would suggest writing a script that loops over the values of k.) How many are misclassified on the testing set for each value?

In [41]:
```python
# Write your answer below
# List of k values
ks = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50]
misclassified_data = {}

# Loop over k values
for k in ks:
    # Initialize the KNeighborsClassifier with the current k
    knn = KNeighborsClassifier(n_neighbors=k)

    # Train the classifier
    knn.fit(X_train, y_train)

    # Predict the labels on the test set
    y_pred = knn.predict(X_test)

    # Calculate the number of misclassified instances
    misclassified = (y_test != y_pred).sum()
    misclassified_data[k] = misclassified

    # Print the results
    print(f"k = {k}: {misclassified} misclassified values")
```

```
k = 1: 9 misclassified values
k = 2: 9 misclassified values
k = 3: 5 misclassified values
k = 4: 6 misclassified values
k = 5: 5 misclassified values
k = 6: 6 misclassified values
k = 7: 5 misclassified values
k = 8: 8 misclassified values
k = 9: 6 misclassified values
k = 10: 8 misclassified values
k = 20: 10 misclassified values
k = 30: 8 misclassified values
k = 40: 7 misclassified values
k = 50: 12 misclassified values
```

6(e) (5 points)

Use random states 10, 11, 12, and 13 and generate 4 more random training/testing sets. Run knn for k = 1,2,3,4,5,6,7,8,9,10,20,30,40,50 on each of these new training/testing sets. For each set, including the original, how many values are misclassified, relate k to the number misclassified.

In [42]:
```python
# Write your answer below
ks = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50]
random_states = [10, 11, 12, 13, 19]  # Adding 19 for the original set

# Dictionary to store misclassified counts for each random state and k value
results = {}

# Loop over random states
for rstate in random_states:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, r
    results[rstate] = {}
```

```
    # Loop over k values
    for k in ks:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        misclassified = (y_test != y_pred).sum()

        results[rstate][k] = misclassified

# Displaying the results
for rstate, data in results.items():
    print(f"Random State: {rstate}")
    for k, misclassified in data.items():
        print(f"\tk = {k}: {misclassified} misclassified values")
    print("\n")
```

```
Random State: 10
        k = 1: 11 misclassified values
        k = 2: 12 misclassified values
        k = 3: 8 misclassified values
        k = 4: 4 misclassified values
        k = 5: 6 misclassified values
        k = 6: 7 misclassified values
        k = 7: 7 misclassified values
        k = 8: 5 misclassified values
        k = 9: 11 misclassified values
        k = 10: 7 misclassified values
        k = 20: 17 misclassified values
        k = 30: 20 misclassified values
        k = 40: 20 misclassified values
        k = 50: 20 misclassified values


Random State: 11
        k = 1: 14 misclassified values
        k = 2: 8 misclassified values
        k = 3: 8 misclassified values
        k = 4: 8 misclassified values
        k = 5: 7 misclassified values
        k = 6: 8 misclassified values
        k = 7: 8 misclassified values
        k = 8: 8 misclassified values
        k = 9: 7 misclassified values
        k = 10: 8 misclassified values
        k = 20: 10 misclassified values
        k = 30: 16 misclassified values
        k = 40: 17 misclassified values
        k = 50: 17 misclassified values


Random State: 12
        k = 1: 9 misclassified values
        k = 2: 12 misclassified values
        k = 3: 8 misclassified values
        k = 4: 10 misclassified values
        k = 5: 10 misclassified values
        k = 6: 10 misclassified values
        k = 7: 8 misclassified values
        k = 8: 8 misclassified values
        k = 9: 6 misclassified values
        k = 10: 6 misclassified values
        k = 20: 7 misclassified values
        k = 30: 7 misclassified values
        k = 40: 12 misclassified values
        k = 50: 14 misclassified values


Random State: 13
        k = 1: 12 misclassified values
        k = 2: 15 misclassified values
        k = 3: 7 misclassified values
        k = 4: 9 misclassified values
        k = 5: 6 misclassified values
        k = 6: 7 misclassified values
        k = 7: 7 misclassified values
        k = 8: 8 misclassified values
```

```
        k = 9: 8 misclassified values
        k = 10: 8 misclassified values
        k = 20: 9 misclassified values
        k = 30: 9 misclassified values
        k = 40: 8 misclassified values
        k = 50: 12 misclassified values


Random State: 19
        k = 1: 9 misclassified values
        k = 2: 9 misclassified values
        k = 3: 5 misclassified values
        k = 4: 6 misclassified values
        k = 5: 5 misclassified values
        k = 6: 6 misclassified values
        k = 7: 5 misclassified values
        k = 8: 8 misclassified values
        k = 9: 6 misclassified values
        k = 10: 8 misclassified values
        k = 20: 10 misclassified values
        k = 30: 8 misclassified values
        k = 40: 7 misclassified values
        k = 50: 12 misclassified values
```