第一题

In [2]:
```
import pandas as pd
titanic = pd.read_csv("insurance.csv")
```

In [4]:
```
# 获取字段名称
cols = titanic.columns.tolist()
print(cols)

# 获取字段数量
col_num = len(cols)
print(col_num)

# 获取样本数量
sam_num = titanic.shape[0]
print(sam_num)
# 获取样本前 5 行样本
five_data = titanic.head()
print(five_data)
```

```
['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']
7
1338
   age     sex     bmi  children smoker     region      charges
0   19  female  27.900         0    yes  southwest  16884.92400
1   18    male  33.770         1     no  southeast   1725.55230
2   28    male  33.000         3     no  southeast   4449.46200
3   33    male  22.705         0     no  northwest  21984.47061
4   32    male  28.880         0     no  northwest   3866.85520
```

第二题

In [5]:
```
import numpy as np
from scipy.stats import stats
test_data = np.random.random(size=100)
# 验证分布
model = stats.normaltest(test_data)
# 输出检验结果
print(model)
```

```
NormaltestResult(statistic=24.19454258664627, pvalue=5.574704239014912e-06)
```

3. 下列属于衡量数据整体散度的是（可多选）：

- a. 欧式距离
- b. 标准差
- c. 分位数
- d. 众数

B、C

第四题

In [6]:
```
import pandas as pd
import numpy as np

example_data = pd.Series([1, 2, 3, np.nan, 4])

# 判断是否含有缺失值
boolean_array = example_data.isnull()
print(boolean_array)

# 缺失值替换
new_data = example_data.fillna('missing')
print(new_data)
```

```
0    False
1    False
2    False
3     True
4    False
dtype: bool
0        1.0
1        2.0
2        3.0
3    missing
4        4.0
dtype: object
```

第五题

In [ ]:
```
import pandas as pd

# 读取数据集
data = pd.read_csv('birthrate.csv')

# 对 'birth rates per capita' 列进行四分位数等频离散化
data_qcut = pd.qcut(data['birth_rates'], q=4, labels=['低', '中低', '中高', '高'

# 显示结果
print(data_qcut)
```

第六题

见本pdf最后

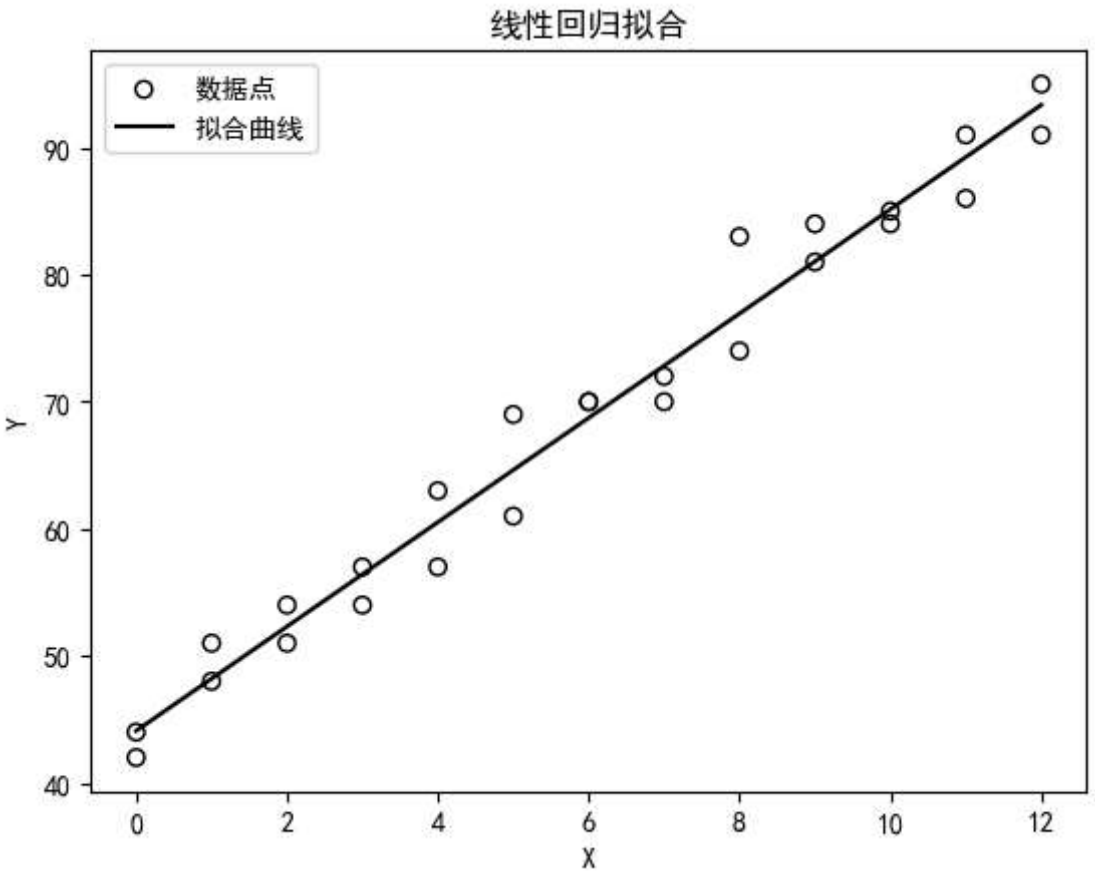
In [ ]:
```

```

第七题

In [137…
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# 给定数据
X = np.array([0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10
Y = np.array([42, 44, 51, 48, 51, 54, 57, 54, 57, 63, 61, 69, 70, 70, 70, 72, 74
```

```
model = LinearRegression()

model.fit(X.reshape(-1, 1), Y)

a = model.intercept_
b = model.coef_[0]
print(f'模型参数：a = {a}, b = {b}')


plt.rcParams['font.family']=['SimHei']#关键是这句
X_fit = np.linspace(X.min(), X.max(), 100)

Y_fit = a + b * X_fit
plt.scatter(X, Y, color='black', label='数据点', facecolors='none', edgecolors='

plt.plot(X_fit, Y_fit, color='black', label='拟合曲线')

plt.legend()

plt.xlabel('X')
plt.ylabel('Y')
plt.title('线性回归拟合')

# 显示图表
plt.show()
```

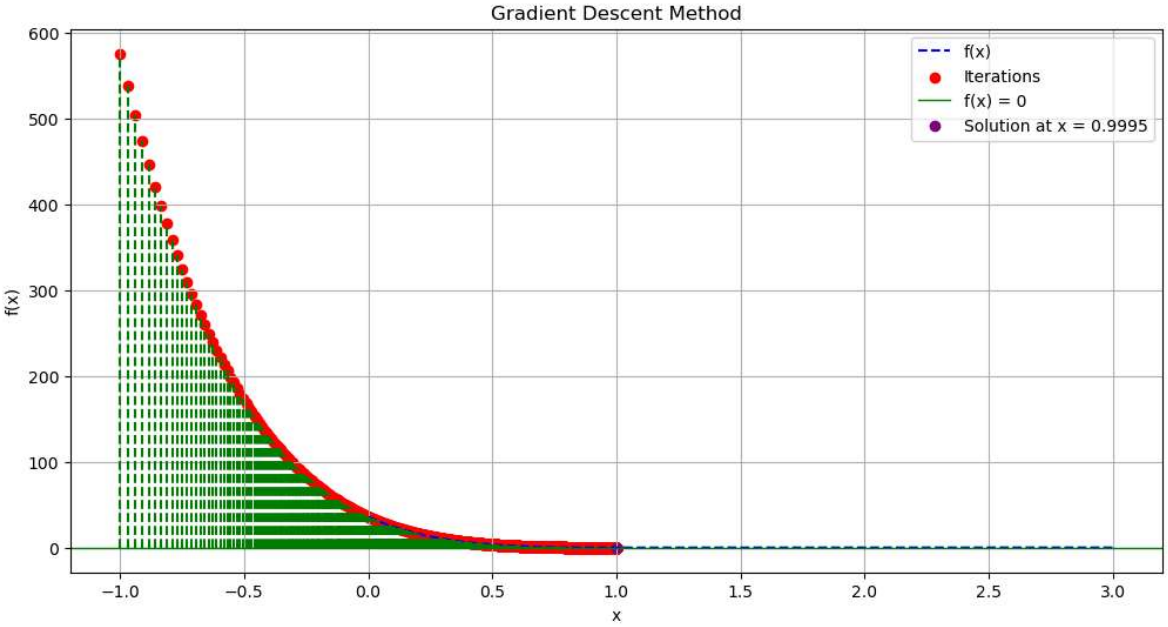
模型参数：a = 44.1043956043956, b = 4.104395604395605



第八题

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
import random

def f(x):
    return (x**3 - 6*x**2 + 11*x - 6)**2

def f_prime(x):
    return (3*x**2 - 12*x + 11)*2*(x**3 - 6*x**2 + 11*x - 6)

def gradient_descent(x0, alpha, max_iter=1000, tol=1e-8):
    x = x0
    x_history = [x]  # 记录x的迭代历史
    f_history = [f(x)]  # 记录f(x)的迭代历史
    for i in range(max_iter):
        grad = f_prime(x)
        x_new = x - alpha * grad
        x_history.append(x_new)
        f_history.append(f(x_new))
        if abs(x_new - x) < tol and abs (f(x_new) < tol):
            break
        x = x_new
    return x, x_history, f_history

x0 = -1
alpha = 0.000025
solution, x_history, f_history = gradient_descent(x0, alpha, max_iter=30000, tol

# 绘制f(x)的图像
x_values = np.linspace(0, 3, 400)
y_values = f(x_values)
plt.figure(figsize=(12, 6))
plt.plot(x_values, y_values, label='f(x)', color='blue', linestyle='--')

# 绘制迭代过程中x的值
plt.scatter(x_history, f_history, color='red', label='Iterations')

# 绘制水平线表示f(x)=0
plt.axhline(0, color='green', linestyle='-', linewidth=1, label='f(x) = 0')

# 绘制迭代过程中的点
for x, fx in zip(x_history, f_history):
    plt.plot([x, x], [0, fx], 'g--')

# 绘制解的位置
plt.scatter([solution], [f(solution)], color='purple', label=f'Solution at x = {

plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent Method')
plt.legend()
plt.grid(True)
plt.show()

print(f'The solution is x = {solution:.4f}')
```




```
The solution is x = 0.9995
```

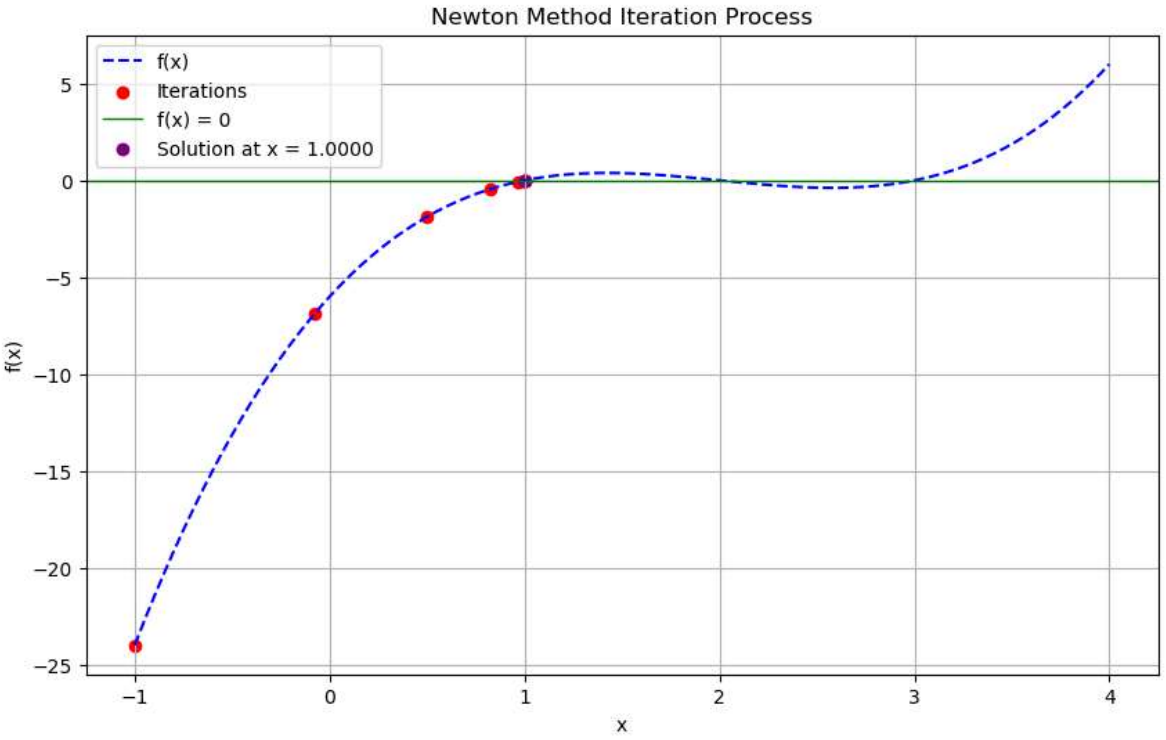
第九题

In [ ]:

In [128…

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x**3 - 6*x**2 + 11*x - 6

def f_prime(x):
    return 3*x**2 - 12*x + 11

def newton_method(x0, max_iter=100, tol=1e-6):
    x = x0
    x_history = [x]  # 记录x的迭代历史
    for i in range(max_iter):
        x_new = x - f(x) / f_prime(x)
        x_history.append(x_new)
        if abs(x_new - x) < tol:
            break
        x = x_new
    return x, x_history

# 选择一个初始猜测值
x0 = -1
solution, x_history = newton_method(x0)

# 绘制迭代过程
plt.figure(figsize=(10, 6))
x_values = np.linspace(-1, 4, 400)
y_values = f(x_values)
plt.plot(x_values, y_values, label='f(x)', color='blue', linestyle='--')

# 绘制迭代点
```

```
f_history = [f(x) for x in x_history]  # 计算每个迭代点的f(x)值
plt.scatter(x_history, f_history, color='red', label='Iterations')

# 绘制水平线表示f(x)=0
plt.axhline(0, color='green', linestyle='-', linewidth=1, label='f(x) = 0')

# 绘制解的位置
plt.scatter([solution], [f(solution)], color='purple', label=f'Solution at x = {

plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Newton Method Iteration Process')
plt.legend()
plt.grid(True)
plt.show()

print(f'The solution is x = {solution:.4f}')
```




```
The solution is x = 1.0000
```

第十题

In [130…

```
import pandas as pd
KEEL = pd.DataFrame({
    'ID': [1, 2, 3, 4, 5],
    'CT': [295.7, 300.0, 290.5, 280.0, 275.3],
    'FA': [98.8, 95.0, 100.2, 102.5, 99.0],
    'WT': [185.6, 190.0, 180.0, 175.5, 170.0],
    'SP': [14.2, 15.0, 13.0, 14.5, 13.5]
})

# 进行 z-score 标准化（不包括 ID 列）
features = data.drop(columns=['ID'])
z_score_standardized = (features - features.mean()) / features.std()

result = pd.concat([data['ID'], z_score_standardized], axis=1)
```

```
# 输出结果
print(result)
```

```
   ID        CT        FA        WT        SP
0   1  0.710112 -0.110133  0.678668  0.201103
1   2  1.122745 -1.505157  1.233712  1.206617
2   3  0.211114  0.403823 -0.027752 -1.307168
3   4 -0.796477  1.248179 -0.595411  0.578171
4   5 -1.247494 -0.036711 -1.289217 -0.678722
```

第六题：

(1) 拟合 $Y=a+bX+\varepsilon$：

$\bar{x}=\frac{2+4}{6}=1$，$\bar{y}=\frac{2+1}{6}=0.5$

则 $b=\frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sum(x_i-\bar{x})^2}$

$=\frac{\sum x_i y_i-n\bar{x}\bar{y}}{\sum x_i^2-n(\bar{x})^2}$

$=\frac{1\times1+1\times2-6\times0.5}{2+4\times2-6\times1}$

$=\frac{0}{2}$

$=0$

$a=\bar{y}-b\cdot\bar{x}=0.5-0=0.5$

∴模型为 $\bar{y}=0.5$

(2) 拟合 $Y=bX+\varepsilon$：

由于 $b=\frac{\sum X_i Y_i}{\sum X_i^2}=\frac{1\times1+1\times2}{1\times2+4\times2}=\frac{3}{10}$

∴模型为 $Y=0.3X$ $=0.3$

```
# 输出结果
print(result)
```

```
   ID        CT        FA        WT        SP
0   1  0.710112 -0.110133  0.678668  0.201103
1   2  1.122745 -1.505157  1.233712  1.206617
2   3  0.211114  0.403823 -0.027752 -1.307168
3   4 -0.796477  1.248179 -0.595411  0.578171
4   5 -1.247494 -0.036711 -1.289217 -0.678722
```

第九题：

① Newton和 Gradient Descent异同：

相同：① 都可用于找函数极值、最小/最大化目标函数

② 都通过 $\theta^k \to \theta^{k+1}$ 的多步迭代方法逼近最优解

差异：① 收敛速度：牛顿迭代在目标函数接近2次时，收敛速度更快，尤其是接近最优解时。梯度下降需要更多次迭代次数逼近最优解，特别是参数空间较大。

② 参数更新：牛顿为 $\theta := \theta - \frac{f(\theta)}{f'(\theta)}$，$\theta := \theta - H^{-1}\nabla_\theta l(\theta)$，牛顿法每次迭代将 H^{-1} 与梯度相乘，并且无需学习率。

梯度下降为 $\theta_j := \theta_j - \alpha \times \frac{\partial}{\partial \theta_j} J(\theta)$，更新与学习率有关。

③ 适用性：梯度下降适用于参数空间大，大规模数据集中。牛顿法适用于参数少，尤其是参数空间平坦，且计算成本大（Hessian 矩阵的逆 H^{-1} 成本高）

牛顿迭代推导见后

```
# 输出结果
print(result)
```

```
   ID        CT        FA        WT        SP
0   1  0.710112 -0.110133  0.678668  0.201103
1   2  1.122745 -1.505157  1.233712  1.206617
2   3  0.211114  0.403823 -0.027752 -1.307168
3   4 -0.796477  1.248179 -0.595411  0.578171
4   5 -1.247494 -0.036711 -1.289217 -0.678722
```

牛顿迭代法公式推导：

将函数 $f(x)$ 在 $x=x_0$ 处泰勒展开：

$$f(x)=f(x_0)+\frac{f'(x_0)}{1!}(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\frac{f'''(x_0)}{3!}(x-x_0)^3+\cdots$$

取等式前2项：

$$f(x)=f(x_0)+\frac{f'(x_0)}{1!}(x-x_0)$$

对等式2边取导数：

$$f'(x)=f'(x_0)+\frac{f''(x_0)}{1!}(x-x_0)$$

即 $f'(x)=f'(x_0)+f''(x_0)\Delta x$

依微积分性质，$f(x)$ 取最小值时，$f'(x)=0$，代入：

$$0=f'(x_0)+f''(x_0)\Delta x$$

即

$$\Delta x=-\frac{f'(x_0)}{f''(x_0)}$$

此时令 $\Delta x=x_{n+1}-x_n$，则：

$$x_{n+1}=x_n-\frac{f'(x_n)}{f''(x_n)}$$