|  |
| --- |
| 数据筛选预处理与其正态分布可视化 |
| 1. def Normal\_distribution\_visulable(ax, DataFrame, Product\_name): 2. translator = Translator() 3. translated\_name = translator.translate(Product\_name, src='zh-CN', dest='en').text 4. filtered\_1 = DataFrame[DataFrame['单品名称'] == Product\_name] 5. mean, std = filtered\_1['批发单价'].mean(), filtered\_1['批发单价'].std() 6. xmin, xmax = filtered\_1['批发单价'].min(), filtered\_1['批发单价'].max() 7. x = np.linspace(xmin, xmax, 100) 8. sns.histplot(filtered\_1['批发单价'], kde=False, bins=30, ax=ax, label='Data') 9. ax.plot(x, norm.pdf(x, mean, std) \* 1000, 'k', linewidth=2, label='Fit') 10. ax.axvline(mean, color='r', linestyle='--', linewidth=2, label='Mean') 11. ax.axvline(mean + 3\*std, color='orange', linestyle='--', linewidth=2, label='+3 Std Dev') 12. ax.axvline(mean - 3\*std, color='orange', linestyle='--', linewidth=2, label='-3 Std Dev') 13. ax.set\_title(translated\_name) 14. ax.set\_xlabel('Value') 15. ax.set\_ylabel('Frequency') 16. ax.legend() 17. def Show\_all\_Normal\_distribution(Dataframe, Length, Width): 18. Number = Length \* Width 19. unique\_values = Dataframe['单品名称'].unique() 20. random\_plt\_name = random.sample(list(unique\_values), Number) 21. fig, axes = plt.subplots(Length, Width, figsize=(4\*Length, 3\*Width)) 22. for ax, name in zip(axes.flatten(), random\_plt\_name): 23. Normal\_distribution\_visulable(ax, Dataframe, name) 24. plt.tight\_layout() 25. plt.show() |

|  |
| --- |
| 肘部法确定最优聚类中心算法 |
| 1. *# 使用 seaborn 设置美观的图表样式* 2. sns.set(style="whitegrid") 3. *# 读入数据* 4. Cluster\_prodect\_raw\_data = pd.read\_excel("副本t1聚类(2).xlsx", engine='openpyxl') 5. *# WCSS 列表* 6. wcss = [] 7. *# 尝试不同数量的聚类中心* 8. for i in range(1, 11): 9. kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0) 10. kmeans.fit(Cluster\_prodect\_raw\_data) 11. wcss.append(kmeans.inertia\_) 12. *# 绘制肘部法图* 13. plt.figure(figsize=(10, 6))  *# 设置图表大小* 14. plt.plot(range(1, 11), wcss, marker='o', linestyle='--')  *# 添加点和线* 15. max\_wcss = max(wcss) 16. plt.fill\_between(range(1, 11), max\_wcss, wcss, color='skyblue', alpha=0.8)  *# 填充天蓝色透明区域在曲线上方* 17. plt.title('Elbow Method', fontsize=16)  *# 设置标题和字体大小* 18. plt.xlabel('Number of clusters', fontsize=14)  *# 设置 x 轴标签和字体大小* 19. plt.ylabel('WCSS', fontsize=14)  *# 设置 y 轴标签和字体大小* 20. plt.xticks(fontsize=12)  *# 设置 x 轴刻度字体大小* 21. plt.yticks(fontsize=12)  *# 设置 y 轴刻度字体大小* 22. plt.show() |

|  |
| --- |
| 蔬菜品类之间斯皮尔曼相关性分析及数据可视化 |
| 1. df = Saled\_KG\_according\_category\_data 2. *# 使用Google Translate API翻译列名* 3. translator = Translator() 4. translated\_columns = [translator.translate(column, src='zh-cn', dest='en').text for column in df.columns] 5. df.columns = translated\_columns 6. *# 计算斯皮尔曼相关性* 7. spearman\_corr = df.corr(method='spearman') 8. *# 创建热力图* 9. plt.figure(figsize=(10, 8))  *# 设置图形大小* 10. sns.set(style="white")  *# 设置风格为白色背景* 11. sns.heatmap(spearman\_corr, annot=True, cmap="coolwarm", square=True, linewidths=.5, cbar\_kws={"shrink": .75}) 12. plt.title('Spearman Correlation Heatmap') 13. plt.show() |

|  |
| --- |
| 利润率-销量线性回归及可视化算法 |
| 1. *# 计算利润率和销售量之间的关系* 2. *# 创建一个2x3的subplot，高为2，宽为3* 3. fig, axs = plt.subplots(2, 3, figsize=(15, 10)) 4. *# 使用一个字典来存储所有的DataFrame，这样我们可以使用一个循环来绘制所有的图* 5. data\_dict = { 6. 'Root Stock': Profit\_KG\_stat\_root\_stock, 7. 'Mosaic Leaf Class': Profit\_KG\_stat\_Mosaic\_leaf\_class, 8. 'Flower Vegetables': Profit\_KG\_stat\_flower\_vegetables, 9. 'Solanaceous': Profit\_KG\_stat\_solanaceous, 10. 'Chili Peppers': Profit\_KG\_stat\_Chili\_peppers, 11. 'Mushrooms': Profit\_KG\_stat\_mushroomss 12. } 13. best\_fit\_equations = {}  *# 用于存储最佳拟合公式* 14. *# 循环通过字典绘制每个图* 15. for i, (key, df) in enumerate(data\_dict.items()): 16. ax = axs[i//3, i%3] 18. *# 提取数据* 19. x = df.iloc[:, 3].values 20. y = df.iloc[:, 1].values 22. *# 创建散点图* 23. ax.scatter(x, y, label='Data Points') 25. min\_mse = float('inf') 26. best\_degree = 0 27. best\_p = None 29. *# 尝试从1到6阶多项式* 30. for degree in range(0,2): 31. z = np.polyfit(x, y, degree) 32. p = np.poly1d(z) 33. mse = mean\_squared\_error(y, p(x)) 35. if mse < min\_mse: 36. min\_mse = mse 37. best\_degree = degree 38. best\_p = p 40. *# 绘制最佳拟合线* 41. xp = np.linspace(min(x), max(x), 100) 42. ax.plot(xp, best\_p(xp), '--',color="r", label=f'Best Fit (Degree {best\_degree})') 44. best\_fit\_equations[key] = best\_p 45. *# 设置图的其他元素* 46. ax.set\_title(f'{key} (Best Degree: {best\_degree})') 47. ax.set\_xlabel('Profit') 48. ax.set\_ylabel('Sell') 49. ax.legend() 50. plt.tight\_layout() 51. plt.show() 52. *# 单独输出最佳拟合公式* 53. print("Best fit equations:") 54. for key, equation in best\_fit\_equations.items(): 55. print(f"{key}: {equation}") |

|  |
| --- |
| 售价-销量线性回归算法及其可视化 |
| 1. *# 创建一个2x3的subplot，高为2，宽为3* 2. fig, axs = plt.subplots(2, 3, figsize=(15, 10)) 3. *# 使用一个字典来存储所有的DataFrame，这样我们可以使用一个循环来绘制所有的图* 4. data\_dict = { 5. 'Root Stock': Profit\_KG\_stat\_root\_stock, 6. 'Mosaic Leaf Class': Profit\_KG\_stat\_Mosaic\_leaf\_class, 7. 'Flower Vegetables': Profit\_KG\_stat\_flower\_vegetables, 8. 'Solanaceous': Profit\_KG\_stat\_solanaceous, 9. 'Chili Peppers': Profit\_KG\_stat\_Chili\_peppers, 10. 'Mushrooms': Profit\_KG\_stat\_mushroomss 11. } 12. best\_fit\_equations = {}  *# 用于存储最佳拟合公式* 13. *# 循环通过字典绘制每个图* 14. for i, (key, df) in enumerate(data\_dict.items()): 15. ax = axs[i//3, i%3] 17. *# 提取数据* 18. x = df.iloc[:, 0].values  *# 假设第一列是利润率* 19. y = df.iloc[:, 1].values  *# 假设第二列是销售量* 21. *# 创建散点图* 22. ax.scatter(x, y, label='Data Points') 24. min\_mse = float('inf') 25. best\_degree = 0 26. best\_p = None 28. *# 尝试从1到6阶多项式* 29. for degree in range(0, 2): 30. z = np.polyfit(x, y, degree) 31. p = np.poly1d(z) 32. mse = mean\_squared\_error(y, p(x)) 34. if mse < min\_mse: 35. min\_mse = mse 36. best\_degree = degree 37. best\_p = p 39. *# 绘制最佳拟合线* 40. xp = np.linspace(min(x), max(x), 100) 41. ax.plot(xp, best\_p(xp), '--',color="r", label=f'Best Fit (Degree {best\_degree})') 43. best\_fit\_equations[key] = best\_p 44. *# 设置图的其他元素* 45. ax.set\_title(f'{key} (Best Degree: {best\_degree})') 46. ax.set\_xlabel('Selling price') 47. ax.set\_ylabel('Sell') 48. ax.legend() 49. plt.tight\_layout() 50. plt.show() 51. *# 单独输出最佳拟合公式* 52. print("Best fit equations:") 53. for key, equation in best\_fit\_equations.items(): 54. print(f"{key}: {equation}") |

|  |
| --- |
| 日期-24节气转换算法 |
| 1. *# 二十四节气及其对应的月和日* 2. solar\_terms = [ 3. ("立春", (2, 4)), ("雨水", (2, 19)), 4. ("惊蛰", (3, 5)), ("春分", (3, 20)), 5. ("清明", (4, 4)), ("谷雨", (4, 20)), 6. ("立夏", (5, 5)), ("小满", (5, 21)), 7. ("芒种", (6, 5)), ("夏至", (6, 21)), 8. ("小暑", (7, 6)), ("大暑", (7, 22)), 9. ("立秋", (8, 7)), ("处暑", (8, 23)), 10. ("白露", (9, 7)), ("秋分", (9, 22)), 11. ("寒露", (10, 8)), ("霜降", (10, 23)), 12. ("立冬", (11, 7)), ("小雪", (11, 22)), 13. ("大雪", (12, 6)), ("冬至", (12, 21)), 14. ("小寒", (1, 5)), ("大寒", (1, 20)) 15. ] 16. *# 季节和节气的映射* 17. season\_to\_terms = { 18. '春': ["立春", "雨水", "惊蛰", "春分", "清明", "谷雨"], 19. '夏': ["立夏", "小满", "芒种", "夏至", "小暑", "大暑"], 20. '秋': ["立秋", "处暑", "白露", "秋分", "寒露", "霜降"], 21. '冬': ["立冬", "小雪", "大雪", "冬至", "小寒", "大寒"] 22. } 23. def find\_previous\_solar\_term(month, day): 24. for i in reversed(range(len(solar\_terms))): 25. term, (term\_month, term\_day) = solar\_terms[i] 26. if (month, day) >= (term\_month, term\_day): 27. return term 28. return None 29. def encode\_solar\_term(month, day): 30. term = find\_previous\_solar\_term(month, day) 31. if term is None: 32. return [0]\*10 34. for season, terms in season\_to\_terms.items(): 35. if term in terms: 36. season\_encoding = [1 if s == season else 0 for s in ['春', '夏', '秋', '冬']] 37. term\_encoding = [1 if t == term else 0 for t in terms] 38. return season\_encoding + term\_encoding 39. return [0]\*10 |

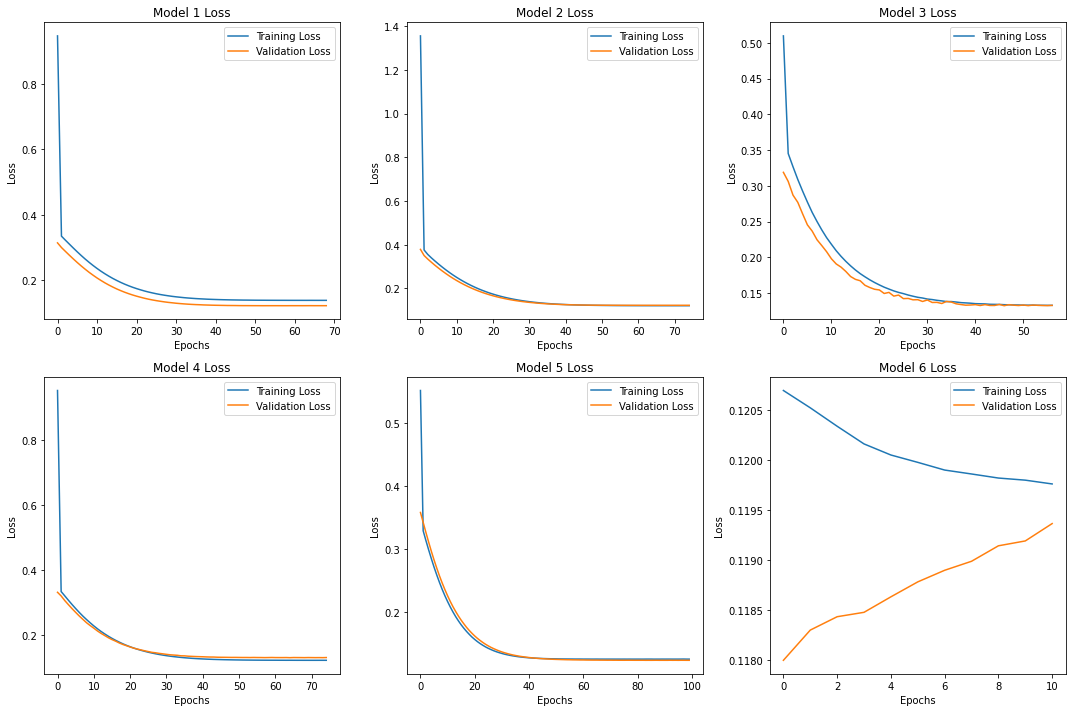
|  |
| --- |
| 滑动窗口算法 |
| 1. def slide\_windows(Dataframe,window\_size): 2. processed\_df = pd.DataFrame() 3. Total\_size = window\_size+7 4. *# 滑动窗口* 5. for i in range(len(Dataframe) - 22): 6. past\_15\_days\_cost = Dataframe.iloc[:, 2].iloc[i:i+window\_size].values 7. future\_7\_days\_solar\_term = np.array(Dataframe.iloc[:, 4].iloc[i+window\_size:i+Total\_size].tolist()) 8. row\_data = { 9. '过去时间窗口成本': list(past\_15\_days\_cost), 10. '未来七天节气': future\_7\_days\_solar\_term.reshape(-1).tolist(),  *# 将7x10的二维数组转换为1D数组* 11. '未来七天成本': list(Dataframe.iloc[:, 2].iloc[i+window\_size:i+Total\_size].values) 12. } 13. processed\_df = processed\_df.append(row\_data, ignore\_index=True) 14. return processed\_df |

|  |
| --- |
| 数据预处理-神经网络训练 |
| 1. def deal\_X2(Dataframe): 2. list\_test = [] 3. for temp in np.array(Dataframe['未来七天节气']): 4. for i in temp: 5. list\_test.append(i) 6. X2 = np.array(list\_test).reshape(-1, 70, 1) 7. return X2 8. def df\_to\_nparray(df\_col): 9. return np.array([np.array(x) for x in df\_col]) 10. def Data\_process\_defore\_train(Dataframe): 11. *# 获取数据集总长度* 12. n = len(Dataframe) 13. *# 计算测试集、验证集和训练集的大小* 14. test\_size = int(n \* 0.2) 15. val\_size = int(n \* 0.8 \* 0.25) 16. train\_size = n - test\_size - val\_size 17. *# 分离自变量和因变量* 18. X = Dataframe[["未来七天节气", "过去时间窗口成本"]] 19. y = Dataframe["未来七天成本"] 20. *# 按照顺序切分数据集* 21. X\_train = X.iloc[:train\_size] 22. y\_train = y.iloc[:train\_size] 23. X\_val = X.iloc[train\_size:train\_size+val\_size] 24. y\_val = y.iloc[train\_size:train\_size+val\_size] 25. X\_test = X.iloc[train\_size+val\_size:] 26. y\_test = y.iloc[train\_size+val\_size:] 27. *# 打印各个数据集的大小* 28. print("Training set:", X\_train.shape, y\_train.shape) 29. print("Validation set:", X\_val.shape, y\_val.shape) 30. print("Test set:", X\_test.shape, y\_test.shape) 31. *# 确保X\_train, y\_train, X\_val, y\_val, X\_test, y\_test都已正确加载并预处理* 32. X1\_train = df\_to\_nparray(X\_train['过去时间窗口成本']).reshape(-1, 15, 1) 33. X2\_train = deal\_X2(X\_train) 34. y\_train\_array = df\_to\_nparray(y\_train).reshape(-1, 7) 35. X1\_val = df\_to\_nparray(X\_val['过去时间窗口成本']).reshape(-1, 15, 1) 36. X2\_val = deal\_X2(X\_val) 37. y\_val\_array = df\_to\_nparray(y\_val).reshape(-1, 7) 38. X1\_test = df\_to\_nparray(X\_test['过去时间窗口成本']).reshape(-1, 15, 1) 39. X2\_test = deal\_X2(X\_test) 40. y\_test\_array = df\_to\_nparray(y\_test).reshape(-1, 7) 41. return {"X1训练集":X1\_train, 42. "X2训练集":X2\_train, 43. "X1验证集":X1\_val, 44. "X2验证集":X2\_val, 45. "X1测试集":X1\_test, 46. "X2测试集":X2\_test, 47. "Y训练集":y\_train\_array, 48. "Y验证集":y\_val\_array, 49. "Y测试集":y\_test\_array} |

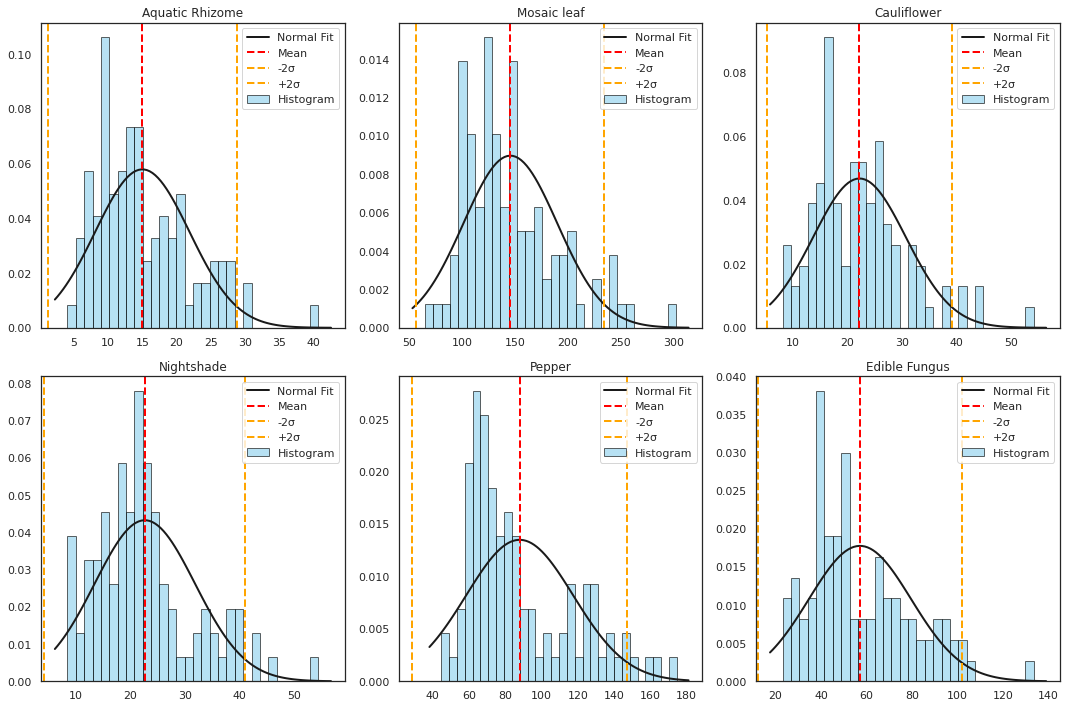
|  |
| --- |
| 特征融合神经网络 |
| 1. *# 注意力机制层* 2. class AttentionLayer(layers.Layer): 3. def \_\_init\_\_(self): 4. super(AttentionLayer, self).\_\_init\_\_() 6. def build(self, input\_shape): 7. self.W = self.add\_weight(shape=(input\_shape[-1], input\_shape[-1]), initializer='random\_normal', trainable=True) 8. self.b = self.add\_weight(shape=(input\_shape[-1],), initializer='zeros', trainable=True) 10. def call(self, inputs): 11. q = tf.nn.tanh(tf.linalg.matmul(inputs, self.W) + self.b) 12. a = tf.nn.softmax(q, axis=1) 13. output = tf.reduce\_sum(a \* inputs, axis=1) 14. return output 15. class Feature\_fusion\_TCN\_model(Model): 16. def \_\_init\_\_(self): 17. super(Feature\_fusion\_TCN\_model, self).\_\_init\_\_() 19. self.tcn1 = TCN() 20. self.tcn2 = TCN() 22. self.concat = layers.Concatenate(axis=1) 23. self.attention\_layer = AttentionLayer() 25. self.reshape = layers.Reshape((1, -1)) 26. self.output\_layer = Dense(7, activation='linear') 28. def call(self, inputs): 29. input1, input2 = inputs 30. tcn1\_output = self.tcn1(input1) 31. tcn2\_output = self.tcn2(input2) 32. merged = self.concat([tcn1\_output, tcn2\_output]) 34. attention\_output = self.attention\_layer(merged) 35. attention\_output\_expanded = self.reshape(attention\_output) 37. output = self.output\_layer(attention\_output\_expanded) 39. return output |

|  |
| --- |
| BFGS最优化 |
| 1. prediction\_functions = [Root\_stock\_prediction, Mosaic\_Leaf\_prediction, Flower\_Vegetables\_prediction, 2. Solanaceous\_prediction, Chili\_Peppers\_prediction, Mushrooms\_prediction] 3. def objective(prices, costs, prediction\_functions): 4. revenue = 0 5. for i in range(len(prices)): 6. quantity = prediction\_functions[i](prices[i]) 7. if quantity < 0: 8. quantity = 0 9. revenue += (prices[i] - costs[i]) \* quantity 10. return -revenue 11. mean\_plus\_2sigma\_values = { 12. '水生根茎类': 28.79, 13. '花叶类': 234.41, 14. '花菜类': 39.24, 15. '茄类': 41.08, 16. '辣椒类': 147.06, 17. '食用菌': 102.06 18. } 19. def optimization\_price(Dataframe, mean\_plus\_2sigma\_values): 20. rows\_price = [] 21. rows\_quantity = [] 22. rows\_profit = [] 23. daily\_optimal\_revenues = []  # 用于存储每天的最优利润 25. for index in range(len(Dataframe)): 26. costs = Dataframe.iloc[index].values 27. initial\_prices = costs \* 1.5 28. max\_prices = [mean\_plus\_2sigma\_values[product] for product in Dataframe.columns] 30. constraints = [ 31. {'type': 'ineq', 'fun': lambda prices, costs=costs: prices - costs}, 32. {'type': 'ineq', 'fun': lambda prices, max\_prices=max\_prices: max\_prices - prices}, 33. {'type': 'ineq', 'fun': lambda prices: [prediction\_functions[i](prices[i]) for i in range(len(prices))]} 34. ] 35. result = minimize(objective, initial\_prices, args=(costs, prediction\_functions), constraints=constraints) 37. if result.success: 38. optimized\_prices = result.x 39. rows\_price.append(optimized\_prices) 41. # 计算当天每个品类的销量和利润 42. quantities = [prediction\_functions[i](optimized\_prices[i]) for i in range(len(optimized\_prices))] 43. profits = [(optimized\_prices[i] - costs[i]) \* quantities[i] for i in range(len(optimized\_prices))] 45. rows\_quantity.append(quantities) 46. rows\_profit.append(profits) 48. # 计算当天的最优利润（注意：我们最小化了目标函数，所以取负数得到最优利润） 49. daily\_optimal\_revenues.append(-result.fun) 51. Result\_price\_dataframe = pd.DataFrame(rows\_price, columns=Dataframe.columns) 52. Result\_quantity\_dataframe = pd.DataFrame(rows\_quantity, columns=Dataframe.columns).applymap(lambda x: 0 if np.abs(x) < 1e-10 else x) 53. Result\_profit\_dataframe = pd.DataFrame(rows\_profit, columns=Dataframe.columns).applymap(lambda x: 0 if np.abs(x) < 1e-10 else x) 55. # 使用Seaborn设置更漂亮的风格 56. sns.set(style="whitegrid") 58. # 可视化每天的最优利润 59. plt.figure(figsize=(8, 7)) 60. plt.bar(range(1, len(daily\_optimal\_revenues) + 1), daily\_optimal\_revenues, color='skyblue', edgecolor='black', alpha=0.6) 61. plt.title("Daily Optimal Revenue") 62. plt.xlabel("Day") 63. plt.ylabel("Optimal Revenue") 64. plt.xticks(range(1, len(daily\_optimal\_revenues) + 1)) 65. plt.show() 66. return Result\_price\_dataframe, Result\_quantity\_dataframe, Result\_profit\_dataframe, daily\_optimal\_revenues 67. # 使用你已经有的 Data\_predicted\_result 68. OPT\_price = optimization\_price(Data\_predicted\_result, mean\_plus\_2sigma\_values) 69. OPT\_price |

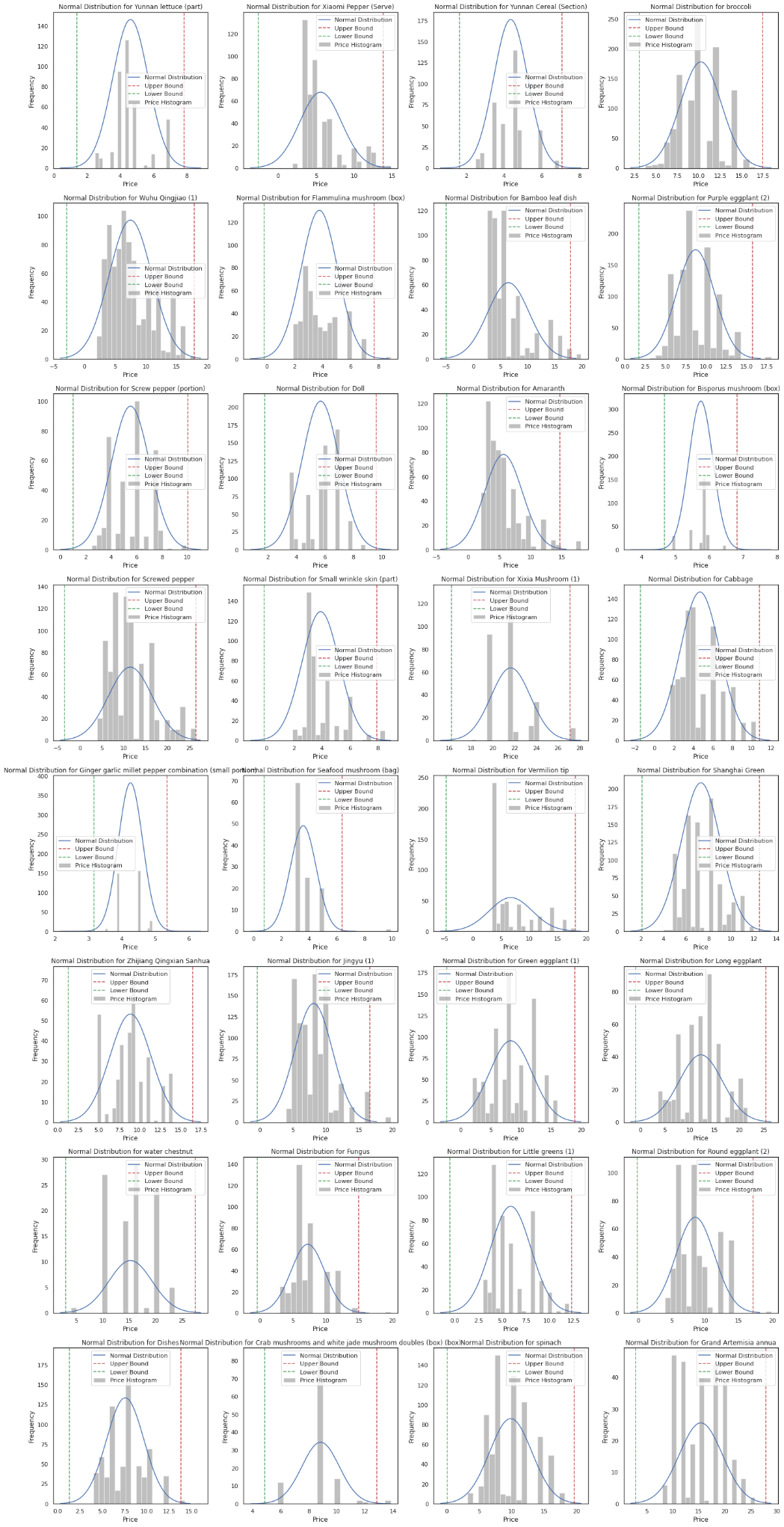
特征融合模型训练过程



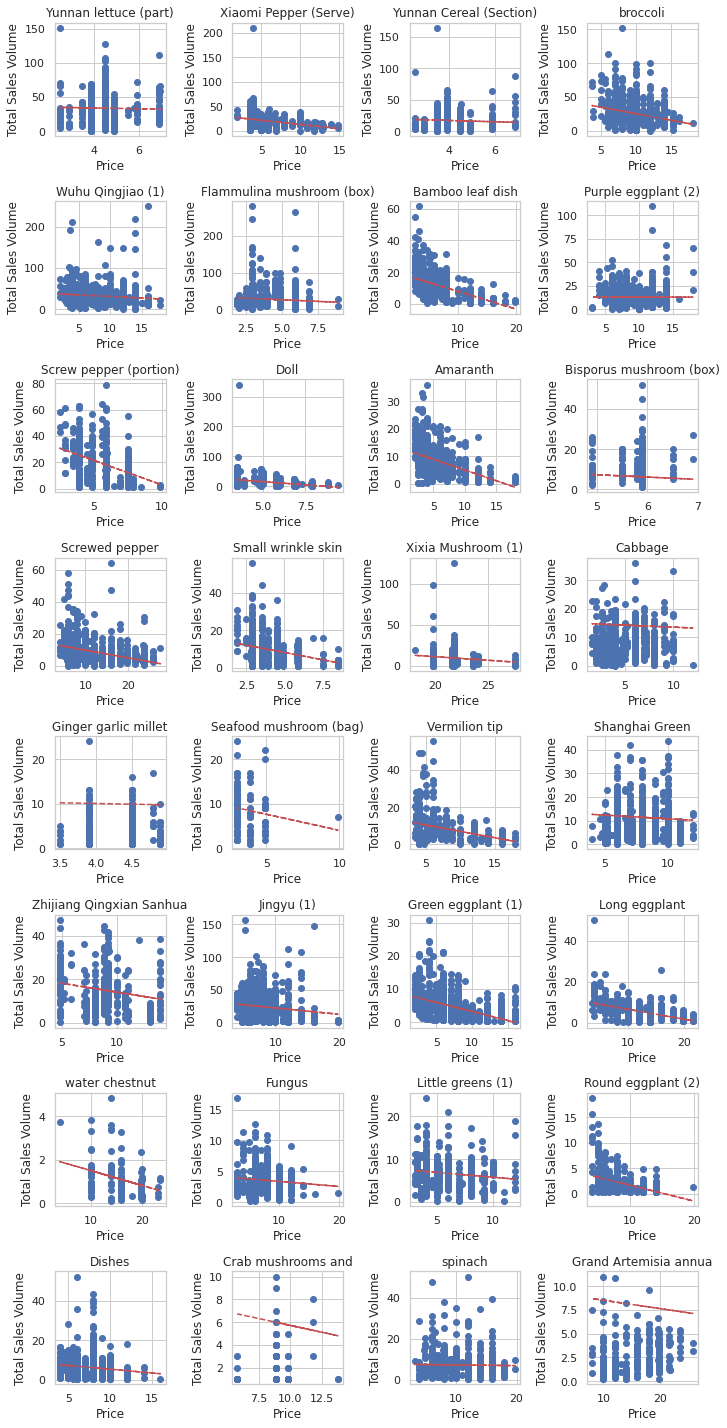
品类Bootstrap sampling



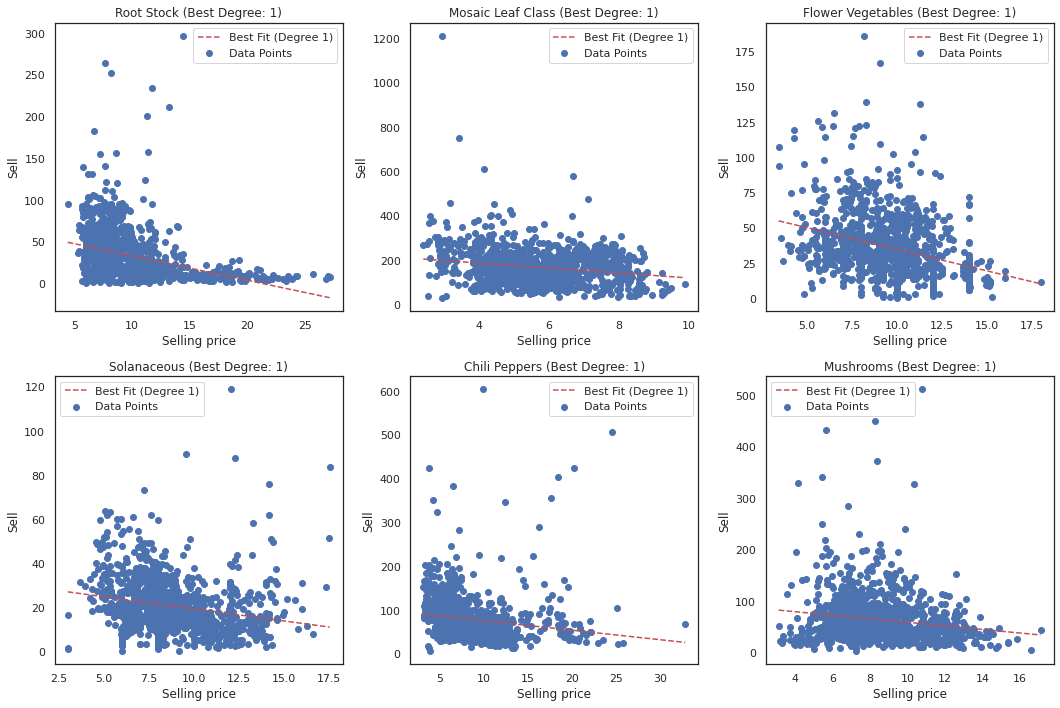
33品类Bootstrap sampling分布



售价-单品销量线性关系图



售价-品类销量线性回归图像



利润率-品类销量线性回归图

