|  |  |
| --- | --- |
| 问题1 python程序 | 数据预处理 |
| #1、缺失值剔除：对某一位点，所有样本都为缺失值则直接剔除该位点  #2、缺失值填补：对某一位点，样本缺失值较多，超过80%，则进行填充  #由于本文已通过0的方式来表现，故不需要主动删除，0即代表删除。  def judge1\_1(df):      c = []      for i in range(354):          a = 0          b = 0          for j in range(40):              if(df.iat[j,i] == 0):                  a += 1              else:                  b = b + df.iat[j,i]          if(a > 20):              for j in range(40):                  df.iat[j,i] = 0              print("第"+str(i)+"列0比较多，故剔除")          else:              for j in range(40):                  if(df.iat[j,i] == 0):                      df.iat[j,i] = b/(40-a)                      c.append(i)      return c  judge1\_1(df2\_5\_1)       #肉眼观测不需要进行缺失值处理  judge1\_1(df2\_5\_2)  #3、最大最小越限处理：根据附件4的最大最小值进行判断  df4\_1 = pd.DataFrame(df4[3])  df4\_1 = df4\_1.drop(df4\_1.index[0])  df4\_1.columns = ["取值范围"]  df4\_1.index = map(str, np.arange(354))  def judge2\_1(df1, df2):      a = 0      b = []      for i in range(0, 354):          ul = re.sub('\(|\)','',df1.iat[i,0])          ul = re.sub('\（|\）','',ul)          if("--" in ul):              upper = "-" + ul.rsplit("-",1)[1]   #上限              lower = ul.rsplit("-",2)[0]         #下限          else:              upper = ul.rsplit("-",1)[1]         #上限              lower = ul.rsplit("-",1)[0]         #下限          upper = float(upper)          lower = float(lower)          for j in range(0, 40):              if(float(df2.iat[j,i]) <= lower or float(df2.iat[j,i]) >= upper):                  df2.iat[j,i] = 0                  b.append(i)      return b  judge2\_1(df4\_1, df2\_5\_1)  judge2\_1(df4\_1, df2\_5\_2)  #4、异常值处理：按照3西格玛原则  def judge2\_2(df3):      a = np.std(df3, axis = 0, ddof =1)      b = np.mean(df3)      c = 0      for i in range(0,354):          for j in range(0,40):              if(np.abs(b[i]-df3.iat[j, i]) > 3\*a[i]):                  df3.iat[j, i] = 0                  c = c+1      return c  #judge2\_2(df2\_5\_1)  judge2\_2(df2\_5\_2) | |

|  |  |
| --- | --- |
| 问题2 python程序 | 降维 |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import MinMaxScaler  from sklearn.preprocessing import StandardScaler  from sklearn.ensemble import RandomForestClassifier  from sklearn.ensemble import RandomForestRegressor  from sklearn.feature\_selection import SelectFromModel  from sklearn.pipeline import Pipeline  df1\_1 = df1\_1.drop(df1\_1.index[0:1])  df1\_1.index = map(str, np.arange(df1\_1.index.size))  column1 = [‘硫含量\_原料’, ‘辛烷值\_原料’, ‘饱和烃’, ‘烯烃’, ‘芳烃’, ‘溴值’, ‘密度’, ‘硫含量\_产品’, ‘辛烷值\_产品’, ‘焦炭\_待生’, ‘硫含量\_待生’, ‘焦炭\_再生’, ‘硫含量\_再生’]  df1\_1.columns = map(str,column1)  df1\_1 = df1\_1.apply(lambda x:x.astype(float))  corr1 = df1\_1.corr(“spearman”)  plt.rcParams[‘font.sans-serif’] = [‘SimHei’]  plt.rcParams[‘axes.unicode\_minus’] = False  sns.set(font=’SimHei’, font\_scale=0.8)  plt.subplots(figsize=(9,9),dpi=1080,facecolor=’w’)  p1 = sns.heatmap(corr1 ,annot=True, vmax=1, square=True, cmap=”Blues”, fmt=’.2g’)  feat\_labels = n1  rf = RandomForestRegressor(n\_estimators=500)  rf.fit(X, y)  importance = rf.feature\_importances\_  #np.argsort()返回待排序集合从下到大的索引值，[::-1]实现倒序，即最终imp\_result内保存的是从大到小的索引值  imp\_result = np.argsort(importance)[::-1][:25]    #按重要性从高到低输出属性列名和其重要性  for i in range(len(imp\_result)):      print("%2d. %-\*s %f" % (i + 1, 30, feat\_labels[imp\_result[i]], importance[imp\_result[i]]))    #对属性列，按属性重要性从高到低进行排序  feat\_labels = [feat\_labels[i] for i in imp\_result]  #绘制特征重要性图像  plt.subplots(dpi=1080)  plt.title('Feature Importance')  plt.bar(range(len(imp\_result)), importance[imp\_result], color='lightblue', align='center')  plt.xticks(range(len(imp\_result)), feat\_labels, rotation=90)  plt.rc\_context({'xtick.color':'black', 'ytick.color':'black'})  plt.xlim([-1, len(imp\_result)])  plt.tight\_layout()  plt.show() | |

|  |  |
| --- | --- |
| 问题3 python程序 | 预测模型 |
| from sklearn.linear\_model import LinearRegression as LR from sklearn.metrics import r2\_score import numpy as np import matplotlib.pyplot as plt import pandas as pd from sklearn.model\_selection import train\_test\_split import sklearn  from sklearn.svm import SVR  import statsmodels.api as sm  from sklearn.ensemble import RandomForestRegressor  #多元线性回归模型 model1 = LR().fit(x\_train, y\_train)  #利用随机森林进行训练 forest = RandomForestRegressor(  n\_estimators=3000,  random\_state=1,  n\_jobs=-1) forest.fit(x\_train, y\_train)  #支持向量回归 linear\_svr = SVR(kernel='linear') model1 = linear\_svr.fit(x\_train, y\_train) linear\_svr\_y\_predict = linear\_svr.predict(x\_test)  # 评价指标 def perfomance\_reg(model,x,y,name=None):  y\_predict = model.predict(x)  check = pd.DataFrame(y)  check['y\_predict'] = y\_predict  check['abs\_err'] = abs(check['y\_predict'] - check[y.name])  check['ape'] = check['abs\_err'] / check[y.name]  ape = check['ape'][check['ape']!=np.inf].mean()  if name:  print(name,':')  print(f'均方误差: {sklearn.metrics.mean\_squared\_error(y,y\_predict)}')  print(f'绝对平均误差: {sklearn.metrics.mean\_absolute\_error(y,y\_predict)}')  print(f'R平方: {r2\_score(y,y\_predict)}')  print(f'平均绝对误差百分比: {ape}')  print('- - - - - - ') perfomance\_reg(model1, x, y)  #可视化 plt.figure() plt.plot(np.arange(len(y\_pred)), y\_test,'go-',label='true value') plt.plot(np.arange(len(y\_pred)),y\_pred,'ro-',label='predict value') plt.title("multipleRegression") plt.legend() # 将样例显示出来 plt.show() | |

|  |  |
| --- | --- |
| 问题4 python程序 | 优化模型 |
| import numpy as np  import pandas as pd  import scipy  import seaborn as sns  import matplotlib.pyplot as plt  import matplotlib as mpl  from random import choices  from turtle import speed  from sklearn.ensemble import RandomForestRegressor  from sklearn.preprocessing import StandardScaler  from sklearn.svm import SVR  from sklearn import preprocessing  #构建预测模型1  x = df3\_1.iloc[:,1:]  y = df3\_1.iloc[:,0]  x = preprocessing.scale(x)#归一化  linear\_svr = SVR(kernel='linear')  linear\_test = linear\_svr.fit(x,y)  #构建预测模型2  y1 = df3\_2.iloc[:,6]  x1 = df3\_2.drop(df3\_2.columns[6], axis = 1)  x1 = preprocessing.scale(x1)#归一化  linear\_svr1 = SVR(kernel='linear')  linear\_test1 = linear\_svr1.fit(x1,y1)  #计算适应度  mpl.rcParams['font.sans-serif'] = ['SimHei']  mpl.rcParams['axes.unicode\_minus'] = False  def fitness\_func(X):      X = np.column\_stack((X\_1, X))      X = preprocessing.scale(X)      return linear\_test.predict(X)  def fitness\_func1(X):      X = np.column\_stack((X\_1\_2, X))      X = preprocessing.scale(X)      return linear\_test1.predict(X)  #更新速度  def velocity\_update(V, X, pbest, gbest, c1, c2, w, max\_val):      size = X.shape[0]      r1 = np.random.random((size, 1))      r2 = np.random.random((size, 1))      V = w\*V+c1\*r1\*(pbest-X)+c2\*r2\*(gbest-X)      # 防止越界处理      for i in range(17):          max = max\_val[i]          min = -max          for j in range(266):              if(V[j,i] > 0):                  V[j,i] = max              elif(V[j,i] < 0):                  V[j,i] = min      return V  #更新位置  def position\_update(X, V):      M = X+V      for i in range(17):          lower = df3\_positon.values[i,0]          upper = df3\_positon.values[i,1]          for j in range(266):              if(M[j,i] > upper):                  M[j,i] = upper              elif(M[j,i] < lower):                  M[j,i] = lower      return M  c1 = 2  c2 = 2  r1 = None  r2 = None  dim = 17  size = 266  iter\_num = 50                      #最大迭代次数  max\_val = df3\_speed.values              #速度范围  fitness\_val\_list = []  f133 = []  s133 = []  # 初始化种群各个粒子的位置  X = X\_2  # 初始化各个粒子的速度  V = df3\_1.iloc[:,8:]  for i in range(dim):      speed1 = df3\_speed.iat[i,0]      speed2 = -speed1      for j in range(size):          V.iat[j,i] = choices([speed2,speed1], [0.5,0.5])[0]  V = V.values  # print(X)  p\_fitness = np.array(y)  g\_fitness = p\_fitness.min()  s\_fitness = fitness\_func1(X)  fitness\_val\_list.append(g\_fitness)  f133.append(p\_fitness[120])  s133.append(s\_fitness[120])  # 初始化的个体最优位置和种群最优位置  pbest = X  gbest = X\_2[129,]  # 迭代计算  for i in range(1, iter\_num):      w = 0.5+(i/iter\_num)\*0.4      V = velocity\_update(V, X, pbest, gbest, c1, c2, w, max\_val)      X = position\_update(X, V)      p\_fitness2 = fitness\_func(X)      s\_fitness2 = fitness\_func1(X)      f133.append(p\_fitness2[120])      s133.append(s\_fitness2[120])      g\_fitness2 = p\_fitness2.min()        # 更新每个粒子的历史最优位置      for j in range(size):          if p\_fitness[j] > p\_fitness2[j]:              pbest[j] = X[j]              p\_fitness[j] = p\_fitness2[j]          # 更新群体的最优位置          if g\_fitness > g\_fitness2:              gbest = X[p\_fitness2.argmin()]              g\_fitness = g\_fitness2          # 记录最优迭代记录          fitness\_val\_list.append(g\_fitness)          i += 1    # 输出迭代结果  print("最优值是：%.5f" % fitness\_val\_list[-1])  print("最优解是：x=%.5f,y=%.5f" % (gbest[0], gbest[1]))  # 绘图  plt.plot(fitness\_val\_list, color='r')  plt.title('迭代过程')  plt.show()  plt.rcParams['font.sans-serif'] = ['SimHei']  plt.rcParams['axes.unicode\_minus'] = False  sns.set(font='SimHei', font\_scale=0.8)  plt.figure(figsize=(20,8))  for n,i in enumerate(range(9,17)):      plt.subplot(2,4,n+1)  #     plt.title(i)      sns.distplot(pbest\_p[i])      plt.xlabel(i, fontsize = 14)      plt.ylabel('')      plt.tick\_params(labelsize = 14) | |

|  |  |
| --- | --- |
| 问题5 python程序 | 可视化 |
| # 绘图  plt.rcParams['font.sans-serif'] = ['SimHei']  # 黑体  plt.rcParams['axes.unicode\_minus'] = False    # 解决无法显示符号的问题  sns.set(font='SimHei', font\_scale=0.8)        # 解决Seaborn中文显示问题  plt.plot(s133, color='r')  plt.title('迭代过程')  plt.show()  # 绘图  plt.rcParams['font.sans-serif'] = ['SimHei']  # 黑体  plt.rcParams['axes.unicode\_minus'] = False    # 解决无法显示符号的问题  sns.set(font='SimHei', font\_scale=0.8)        # 解决Seaborn中文显示问题  plt.plot(f133\_1, color='r')  plt.title('迭代过程')  plt.show() | |