```python
from __future__ import division
from pandas import read_csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout
import sklearn.model_selection
from sklearn.preprocessing import MinMaxScaler
from collections import Counter


def detect_outliers(df, n, features):
    outlier_indices = []
    # iterate over features(columns)
    for col in features:
        # 1st quartile (25%)
        Q1 = np.percentile(df[col], 25)
        # 3rd quartile (75%)
        Q3 = np.percentile(df[col], 75)
        # Interquartile range (IQR)
        IQR = Q3 - Q1
        # outlier step
        outlier_step = 1.5 * IQR
        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 +
outlier_step)].index
        # append the found outlier indices for col to the list of outlier
indices
        outlier_indices.extend(outlier_list_col)
    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(k for k, v in outlier_indices.items() if v > n)
    return multiple_outliers


#  数据预处理
f = open('1.csv', encoding='UTF-8')
names = ['Date', 'OLR', 'F', 'inCOD', 'HLR', 'ALR', 'pH0', 'T', 'TSS', 'VFA',
'outCOD', 'CODrate', 'VFA1', 'pH1', 'T_in', 'T_high', 'T_low']
data = read_csv(f, names=names)

plt.scatter(data['Date'], data['VFA1'])
plt.show()

dataframe = pd.DataFrame(data)
dataframe.boxplot()
#  找到缺失值所在列
tmp = dataframe.isnull().any()
print(tmp)

#  中位数填充缺失值
dataframe['T_high'].fillna(dataframe['T_high'].mean(), inplace=True)

tmp2 = dataframe.isnull().any()
```

```python
print(tmp2)

# detect outliers
Outliers_to_drop = detect_outliers(dataframe, 2, ['OLR', 'F', 'inCOD', 'HLR',
'ALR', 'pH0', 'T', 'TSS', 'VFA', 'outCOD',
                                                  'CODrate', 'VFA1', 'pH1',
'T_in', 'T_high', 'T_low'])
print(Outliers_to_drop)
# Drop outliers

dataframe = dataframe.drop(Outliers_to_drop, axis=0).reset_index(drop=True)

dataframe.to_csv('result1.csv', index=False, sep=',')

#  线性相关性度量
print(dataframe.corr())
dataframe.corr().to_csv('corr1.csv', index=False, sep=',')

#  读取新的数据集文件，进行后续处理
file = open('result1.csv', encoding='UTF-8')


data2 = read_csv(file)
print(data2)
print(data.shape)

#  获取属性
x = dataframe.loc[:, ['F', 'inCOD', 'VFA', 'T_in', 'pH1', 'TSS']]
print(x)
print(x.shape)  # (365,6)

#  以下标形式获取数据（此处为VFA1）
y = dataframe.loc[:, ['VFA1']]
print(y.shape)  # (365,1)
print(y)

#  切分数据集（80%训练集、20%测试集）不过数据量有点小，可能要换方式切分数据集
x_train, x_valid, y_train, y_valid = sklearn.model_selection.train_test_split(x,
y, test_size=0.2)


print(x_train.shape)
print(y_train.shape)
print(x_valid.shape)
print(y_valid.shape)

# 转成DataFrame格式方便数据处理
x_train_pd = pd.DataFrame(x_train)
y_train_pd = pd.DataFrame(y_train)
x_valid_pd = pd.DataFrame(x_valid)
y_valid_pd = pd.DataFrame(y_valid)

# 训练集归一化
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(x_train_pd)
x_train = min_max_scaler.transform(x_train_pd)
print(x_train)
```

```
min_max_scaler.fit(y_train_pd)
y_train = min_max_scaler.transform(y_train_pd)

# 验证集归一化
min_max_scaler.fit(x_valid_pd)
x_valid = min_max_scaler.transform(x_valid_pd)

min_max_scaler.fit(y_valid_pd)
y_valid = min_max_scaler.transform(y_valid_pd)
print(y_valid)


model = Sequential()  # 初始化
model.add(Dense(units = 10,   # 输入大小
                activation='relu',  # 激励函数
                input_shape=(x_train_pd.shape[1],)  # 输入大小, 也就是列的大小
            )
        )

model.add(Dropout(0.2))

model.add(Dense(units=8,
                activation='relu'  # 激励函数
            )
        )

model.add(Dense(units=1,
                activation='linear'  # 线性激励函数 回归一般在输出层用这个激励函数
            )
        )

# print(model.summary())  # 打印网络层次结构

model.compile(loss='mse',  # 损失均方误差
            optimizer='adam',  # 优化器
            )

model.fit(x_train, y_train,
        epochs=1000,  # 迭代次数
        batch_size=128,  # 每次用来梯度下降的批处理数据大小
        shuffle=True,
        verbose=2,
        validation_data = (x_valid, y_valid)  # 验证集
    )

# 预测
y_new = model.predict(x_valid)

# 反归一化
min_max_scaler.fit(y_valid_pd)
y_new = min_max_scaler.inverse_transform(y_new)
y_valid = min_max_scaler.inverse_transform(y_valid)

print(y_new)
print(y_new.shape)
print(y_valid)
print('---')
print(y_valid.shape)
```

```
sum = 0
for i in range(73):
    if abs(y_new[i][0] - y_valid[i][0]) < 0.2*y_valid[i][0]:
        sum = sum + 1

print(sum/73)
```