# 2025年实习项目过程文档

## 项目名称

基于PyQt5的文件管理与批量处理系统开发

## 项目简介

本项目旨在设计并开发一套桌面级文件管理系统，支持文件的批量复制、移动、删除、重命名等基本操作，同时提供简洁的图形用户界面（GUI）和操作日志记录功能。项目采用PyQt5进行界面开发，SQLite数据库进行日志存储，旨在提升文件管理效率与操作规范性。

## 项目时间线

- 2025年3月初：项目需求分析与技术选型  
- 2025年3月中：环境搭建与PyQt5界面原型设计  
- 2025年3月底：实现文件选择与基本操作功能  
- 2025年4月初：添加操作日志记录模块  
- 2025年4月中：功能测试与界面优化  
- 2025年4月末：项目总结与文档编写

## 我的主要工作

系统需求分析与功能设计，PyQt5界面设计与实现，文件操作功能开发，SQLite数据库日志模块实现，系统测试与优化。

# 项目详细过程与源代码

## 1. 配置模块（config.py）

负责存放全局变量，例如数据库名称等，方便后期维护。

# config.py  
  
DATABASE\_NAME = 'file\_log.db'

## 2. 数据库管理模块（db\_manager.py）

数据库操作模块，负责建表、插入日志、查询日志等功能。

# db\_manager.py  
  
import sqlite3  
from config import DATABASE\_NAME  
  
class DBManager:  
 def \_\_init\_\_(self):  
 self.conn = sqlite3.connect(DATABASE\_NAME)  
 self.create\_table()  
  
 def create\_table(self):  
 cursor = self.conn.cursor()  
 cursor.execute('''  
 CREATE TABLE IF NOT EXISTS logs (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 operation TEXT,  
 file\_path TEXT,  
 result TEXT,  
 timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP  
 )  
 ''')  
 self.conn.commit()  
  
 def insert\_log(self, operation, file\_path, result):  
 cursor = self.conn.cursor()  
 cursor.execute('INSERT INTO logs (operation, file\_path, result) VALUES (?, ?, ?)', (operation, file\_path, result))  
 self.conn.commit()  
  
 def get\_logs(self):  
 cursor = self.conn.cursor()  
 cursor.execute('SELECT \* FROM logs ORDER BY timestamp DESC')  
 return cursor.fetchall()  
  
 def close(self):  
 self.conn.close()

## 3. 日志模块（logger.py）

日志模块，调用数据库管理模块，进行统一的日志记录。

# logger.py  
  
from db\_manager import DBManager  
  
class Logger:  
 def \_\_init\_\_(self):  
 self.db = DBManager()  
  
 def log(self, operation, file\_path, result='成功'):  
 self.db.insert\_log(operation, file\_path, result)  
  
 def fetch\_logs(self):  
 return self.db.get\_logs()  
  
 def close(self):  
 self.db.close()

## 4. 主界面程序（main.py）

主程序，搭建PyQt5界面并实现核心文件管理功能。

# main.py  
  
import sys  
import shutil  
import os  
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QFileDialog, QVBoxLayout, QListWidget, QMessageBox  
from logger import Logger  
  
class FileManager(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.init\_ui()  
 self.files = []  
 self.logger = Logger()  
  
 def init\_ui(self):  
 self.setWindowTitle('文件管理系统')  
 layout = QVBoxLayout()  
  
 self.file\_list = QListWidget()  
 layout.addWidget(self.file\_list)  
  
 self.select\_button = QPushButton('选择文件')  
 self.select\_button.clicked.connect(self.select\_files)  
 layout.addWidget(self.select\_button)  
  
 self.copy\_button = QPushButton('复制文件')  
 self.copy\_button.clicked.connect(self.copy\_files)  
 layout.addWidget(self.copy\_button)  
  
 self.move\_button = QPushButton('移动文件')  
 self.move\_button.clicked.connect(self.move\_files)  
 layout.addWidget(self.move\_button)  
  
 self.delete\_button = QPushButton('删除文件')  
 self.delete\_button.clicked.connect(self.delete\_files)  
 layout.addWidget(self.delete\_button)  
  
 self.setLayout(layout)  
  
 def select\_files(self):  
 files, \_ = QFileDialog.getOpenFileNames(self, '选择文件')  
 if files:  
 self.files = files  
 self.file\_list.clear()  
 self.file\_list.addItems(self.files)  
  
 def copy\_files(self):  
 target\_dir = QFileDialog.getExistingDirectory(self, '选择目标文件夹')  
 if target\_dir:  
 for file in self.files:  
 try:  
 shutil.copy(file, target\_dir)  
 self.logger.log('复制', file, '成功')  
 except Exception as e:  
 self.logger.log('复制', file, f'失败: {e}')  
 QMessageBox.information(self, '完成', '文件复制完成')  
  
 def move\_files(self):  
 target\_dir = QFileDialog.getExistingDirectory(self, '选择目标文件夹')  
 if target\_dir:  
 for file in self.files:  
 try:  
 shutil.move(file, target\_dir)  
 self.logger.log('移动', file, '成功')  
 except Exception as e:  
 self.logger.log('移动', file, f'失败: {e}')  
 QMessageBox.information(self, '完成', '文件移动完成')  
  
 def delete\_files(self):  
 reply = QMessageBox.question(self, '确认删除', '确定要删除选中的文件吗？', QMessageBox.Yes | QMessageBox.No, QMessageBox.No)  
 if reply == QMessageBox.Yes:  
 for file in self.files:  
 try:  
 os.remove(file)  
 self.logger.log('删除', file, '成功')  
 except Exception as e:  
 self.logger.log('删除', file, f'失败: {e}')  
 QMessageBox.information(self, '完成', '文件删除完成')  
  
 def closeEvent(self, event):  
 self.logger.close()  
 event.accept()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 fm = FileManager()  
 fm.show()  
 sys.exit(app.exec\_())

## 总结

本项目通过模块化开发方式，提升了文件管理系统的可维护性和扩展性，也加深了我对PyQt5界面开发和数据库应用的理解。

## 附录

    def showVideo(self):

        # 先处理，得到结果视频

        if not self.video\_path:

            print('请先选择视频，再进行操作！')

            return

        conf2 = self.conf2.value()

        conf2 = float("{:.2f}".format(conf2))

        print("Current value:", conf2)

        print(type(conf2))

        IOU2 = self.IOU2.value()

        IOU2 = float("{:.2f}".format(IOU2))

        # 看是否转换的格式符合要求

        print(IOU2)

        if self.class2.text() == '':

            class2 = -1

        else:

            class2 = int(self.class2.text())

        # YOLOv8 - img  start

        # 根据用户combo选择加载预训练模型

        if self.model2.currentText() == '物体检测':

            model = YOLO('yolov8n.pt')

            # model = YOLO('E:\YOLOv8\_物体分类检测\\train\_object\\runs\detect\\train10\weights\\best.pt')

        elif self.model2.currentText() == '实例分割':

            model = YOLO('yolov8n-seg.pt')

        # 打开视频文件

        cap = cv2.VideoCapture(self.video\_path)

        # 获取视频帧的维度

        frame\_width = int(cap.get(3))

        frame\_height = int(cap.get(4))

        # 拿到当前图片路径末尾的文件名

        file\_name = os.path.basename(self.video\_path)

        # 创建VideoWriter对象，指定输出视频目录

        fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

        out = cv2.VideoWriter("video\_output/" + file\_name, fourcc, 20.0,

                              (frame\_width, frame\_height))

        # 循环视频帧

        while cap.isOpened():

            # 读取某一帧

            success, frame = cap.read()

            if success:

                # 用户没有指定classes类别

                if class2 == -1:

                    results = model.predict(frame, conf=conf2, iou=IOU2)

                # 用户指定了类别

                else:

                    results = model.predict(frame, conf=conf2, iou=IOU2, classes=class2)

                # 可视化结果

                annotated\_frame = results[0].plot()

                # 将带注释的帧写入视频文件

                out.write(annotated\_frame)

            else:

                # 最后结尾中断视频帧循环

                break

        # 释放读取和写入对象

        cap.release()

        out.release()

        # 播放已经处理好的物体检测视频

        try:

            self.media\_player2.setMedia(QMediaContent(QUrl("./video\_output/" + file\_name)))

            self.media\_player2.play()

            print("看看预测好的file\_path" + "./video\_output/" + file\_name)

        except Exception as e:

            print(e)

    # 播放/暂停

    def playPause2(self):

        if self.media\_player2.state() == 1:

            self.media\_player2.pause()

        else:

            self.media\_player2.play()

    # 视频总时长获取

    def getDuration2(self, d):

        # d是获取到的视频总时长（ms）

        self.ui.slider2.setRange(0, d)

        self.ui.slider2.setEnabled(True)

        self.displayTime2(d)

    # 视频实时位置获取

    def getPosition2(self, p):

        self.ui.slider2.setValue(p)

        self.displayTime2(self.ui.slider2.maximum() - p)

    # 显示剩余时间

    def displayTime2(self, ms):

        minutes = int(ms / 60000)

        seconds = int((ms - minutes \* 60000) / 1000)

        self.ui.time2.setText('{}:{}'.format(minutes, seconds))

    # 用进度条更新视频位置

    def updatePosition2(self, v):

        self.media\_player2.setPosition(v)

        self.displayTime2(self.ui.slider2.maximum() - v)

    # -- tab3 track 点击事件回调函数 ---

    def chooseVideo3(self):

        try:

            # 拿到视频路径，存到track\_path里，并在textBrowser中展示路径

            file\_dialog = QFileDialog()

            file\_path, \_ = file\_dialog.getOpenFileName(self, 'Open Video File', '', 'Videos (\*.mp4 \*.avi)')

            if file\_path:

                # self.track\_path = file\_path

                print("file\_path", file\_path)

                # 展示路径

                self.track\_path.setText(file\_path)

        except Exception as e:

            print(e)

    # 播放/暂停

    def playPause3(self):

        if self.media\_player3.state() == 1:

            self.media\_player3.pause()

        else:

            self.media\_player3.play()

    # 视频总时长获取

    def getDuration3(self, d):

        # d是获取到的视频总时长（ms）

        self.ui.slider3.setRange(0, d)

        self.ui.slider3.setEnabled(True)

        self.displayTime3(d)

    # 视频实时位置获取

    def getPosition3(self, p):

        self.ui.slider3.setValue(p)

        self.displayTime3(self.ui.slider3.maximum() - p)

    # 显示剩余时间

    def displayTime3(self, ms):

        minutes = int(ms / 60000)

        seconds = int((ms - minutes \* 60000) / 1000)

        self.ui.time3.setText('{}:{}'.format(minutes, seconds))

    # 用进度条更新视频位置

    def updatePosition3(self, v):

        self.media\_player3.setPosition(v)

        self.displayTime3(self.ui.slider3.maximum() - v)

    # 物体追踪，轨迹识别与绘制

    def showTrack(self):

        # 创建一个字典，记录位置信息，格式如下

        # id1 [{x1,y1}{x2,y2}...]

        # id2 [{x1,y1}{x2,y2}...]

        self.locations = {}

        # 先处理，得到结果视频

        if not self.track\_path:

            return

        # 拿到用户输入的参数

        conf3 = self.conf3.value()

        conf3 = float("{:.2f}".format(conf3))

        # 看是否转换的格式符合要求

        print(conf3)

        IOU3 = self.IOU3.value()

        IOU3 = float("{:.2f}".format(IOU3))

        if self.class3.text() == '':

            class3 = -1

        else:

            class3 = int(self.class3.text())

        try:

            # 加载 YOLOv8 model

            model = YOLO('yolov8n.pt')

            # 或者加载自己训练好的模型

            # model = YOLO('E:\YOLOv8\_物体分类检测\\train\_object\\runs\detect\\train10\weights\\best.pt')

            # 实例分割模型

            # model = YOLO('yolov8n-seg.pt')

            # 拿到当前图片路径末尾的文件名

            file\_name = os.path.basename(self.track\_path.toPlainText())

            # Open the video file

            video\_path = 'video/' + file\_name

            print("视频路径：", video\_path)

            cap = cv2.VideoCapture(video\_path)

            # 存储追踪信息

            track\_history = defaultdict(lambda: [])

            # 创建一个列表，用于存储每一帧的图像和跟踪结果

            video\_frames = []

            # 创建一个字典，储存对应类别检测到的物体数量

            category\_nums = {}

            # visited[] 数组记录track\_id，防止重复计数

            visited = set()

            # flag标记

            flag = False

            # video frames 循环每一帧

            while cap.isOpened():

                # 读取一帧

                success, frame = cap.read()

                if success:

                    # 用户没有指定类别

                    if class3 == -1:

                        print("不指定类别")

                        # Run YOLOv8 tracking 模型

                        results = model.track(frame, persist=True, conf=conf3, iou=IOU3, show=False)

                    # 用户指定了类别

                    else:

                        print("指定类别")

                        # Run YOLOv8 tracking 模型

                        results = model.track(frame, persist=True, conf=conf3, iou=IOU3, classes=class3, show=False)

                    # 提取结果中的位置、类别信息

                    boxes = results[0].boxes.xywh.cpu()

                    classes = results[0].boxes.cls.cpu()

                    print("boxes:", boxes)

                    print("classes:", classes)

                    print("results[0].boxes.id:", results[0].boxes.id)

                    if results[0].boxes.id is not None:

                        track\_ids = results[0].boxes.id.int().cpu().tolist()

                        # 绘制结果到框frame中

                        annotated\_frame = results[0].plot()

                        # 拿到 boxes这个矩形框 和 track跟踪的IDs 以及 预测的类别

                        # 根据不同车不同锚框来绘制轨迹

                        for box, track\_id, category in zip(boxes, track\_ids, classes):

                            x, y, w, h = box

                            # 将位置信息存入字典里

                            if track\_id not in self.locations:

                                self.locations[track\_id] = []

                            self.locations[track\_id].append({"x": float(x), "y": float(y), "c": int(category)})

                            # 将类别计数存到字典里

                            # 搞个标志 visited() 集合，如果track\_id已经遍历过了，就将id放入visited里，做个标志

                            flag = True

                            for i in visited:

                                if track\_id == i:

                                    flag = False

                                    break

                            if flag:

                                if int(category) in category\_nums:

                                    category\_nums[int(category)] += 1

                                else:

                                    category\_nums[int(category)] = 1

                                print(category\_nums)

                            visited.add(track\_id)

                            track = track\_history[track\_id]  # 不同车对应不同track

                            track.append((float(x), float(y)))  # x, y 锚框中心点坐标

                            # 超过100帧前的轨迹段会消失

                            if len(track) > 150:

                                track.pop(0)  # 弹出队列最前面的元素

                            # 绘制交通轨迹

                            points = np.hstack(track).astype(np.int32).reshape((-1, 1, 2))

                            cv2.polylines(annotated\_frame, [points], isClosed=False, color=(250, 170, 0), thickness=8)

                        # 将带有跟踪结果的帧添加到帧列表中

                        video\_frames.append(annotated\_frame)

                    else:

                        print("没有id（物体）也给我补帧，不要抽帧")

                        video\_frames.append(frame)

                else:

                    # Break the 循环 if video 到达末尾

                    break

            # 获取视频帧的维度

            frame\_width = int(cap.get(3))

            frame\_height = int(cap.get(4))

            # 创建VideoWriter对象

            fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

            out = cv2.VideoWriter("video\_output/" + file\_name, fourcc, 20.0,

                                  (frame\_width, frame\_height))

            # 遍历带轨迹的帧列表，合成一起组成视频并写入到目标视频中

            for frame in video\_frames:

                # Write the annotated frame to the output video

                out.write(frame)

            # 释放摄像头流

            cap.release()

            # 关闭输出视频流

            out.release()

            # 词频加到self变全局，页面6会用到

            self.category\_nums = category\_nums

            print('category\_nums：', self.category\_nums)

            if hasattr(self, 'categoty\_nums'):

                print('存在')

        except Exception as e:

            print(e)

        # 处理好后，展示轨迹视频

        try:

            self.media\_player3.setMedia(QMediaContent(QUrl("./video\_output/" + file\_name)))

            self.media\_player3.play()

            print("看看预测好的file\_path" + "./video\_output/" + file\_name)

        except Exception as e:

            print(e)

    # 导出位置坐标对话框

    def export\_location(self):

        file\_path, \_ = QFileDialog.getSaveFileName(self, 'Save Chart Data', '', 'Chart (\*.xlsx \*.csv)')

        if file\_path:

            file\_name = os.path.splitext(file\_path)[0]

            print("文件名为:", file\_name)

            file\_extension = os.path.splitext(file\_path)[1]

            print("文件后缀为:", file\_extension)

            # 将数据导出为xlsx

            if file\_extension == '.xlsx':

                # 创建一个空的DataFrame

                df = pd.DataFrame(columns=['Track ID', 'X', 'Y', 'Category'])

                # 遍历字典，并将嵌套位置信息展平后添加到DataFrame中

                for track\_id, positions in self.locations.items():

                    for position in positions:

                        # 保留小数点后3位

                        X = round(position['x'], 3)

                        Y = round(position['y'], 3)

                        df = df.\_append({'Track ID': track\_id, 'X': X, 'Y': Y, 'Category': position['c']},

                                        ignore\_index=True)

                # 将DataFrame写入xlsx文件

                df.to\_excel(file\_path, index=False)

            # 将数据导出为csv

            else:

                # 将数据写入csv文件

                with open(file\_path, 'w', newline='') as csvfile:

                    writer = csv.writer(csvfile)

                    writer.writerow(['Track ID', 'X', 'Y', 'Category'])

                    for track\_id, positions in self.locations.items():

                        for position in positions:

                            # 保留小数点后3位

                            X = round(position['x'], 3)

                            Y = round(position['y'], 3)

                            writer.writerow([track\_id, X, Y, position['c']])

        else:

            print('用户取消了保存图片操作')

            return

    # --- tab4 count 点击事件回调函数 ---

    def chooseVideo4(self):

        try:

            # 拿到视频路径，存到track\_path里，并在textBrowser中展示路径

            file\_dialog = QFileDialog()

            file\_path, \_ = file\_dialog.getOpenFileName(self, 'Open Video File', '', 'Videos (\*.mp4 \*.avi)')

            if file\_path:

                # self.track\_path = file\_path

                print("file\_path", file\_path)

                # 展示路径

                self.count\_path.setText(file\_path)

        except Exception as e:

            print(e)

    # 播放/暂停

    def playPause4(self):

        if self.media\_player4.state() == 1:

            self.media\_player4.pause()

        else:

            self.media\_player4.play()

    # 视频总时长获取

    def getDuration4(self, d):

        # d是获取到的视频总时长（ms）

        self.ui.slider4.setRange(0, d)

        self.ui.slider4.setEnabled(True)

        self.displayTime4(d)

    # 视频实时位置获取

    def getPosition4(self, p):

        self.ui.slider4.setValue(p)

        self.displayTime4(self.ui.slider4.maximum() - p)

    # 显示剩余时间

    def displayTime4(self, ms):

        minutes = int(ms / 60000)

        seconds = int((ms - minutes \* 60000) / 1000)

        self.ui.time4.setText('{}:{}'.format(minutes, seconds))

    # 用进度条更新视频位置

    def updatePosition4(self, v):

        self.media\_player4.setPosition(v)

        self.displayTime4(self.ui.slider4.maximum() - v)

    # 关键步骤 - 计数

    def showCount(self):

        # 先处理，得到结果视频

        if not self.count\_path:

            return

        # 拿到用户输入的参数

        conf4 = self.conf4.value()

        conf4 = float("{:.2f}".format(conf4))

        IOU4 = self.IOU4.value()

        IOU4 = float("{:.2f}".format(IOU4))

        if self.class4.text() == '':

            class4 = -1

        else:

            class4 = int(self.class4.text())

        try:

            # 加载 YOLOv8 model

            model = YOLO('yolov8n.pt')

            # 实例分割模型

            # model = YOLO('yolov8n-seg.pt')

            # 拿到当前图片路径末尾的文件名

            file\_name = os.path.basename(self.count\_path.toPlainText())

            # Open the video file

            video\_path = 'video/' + file\_name

            cap = cv2.VideoCapture(video\_path)

            # Store the track history

            track\_history = defaultdict(lambda: [])

            # 创建一个列表，用于存储每一帧的图像和跟踪结果

            video\_frames = []

            # 定义越线位置（示例，您需要根据实际情况调整）

            line\_position = 600  # 纵坐标 y，表示线段的位置

            # 初始化越线计数器和记录已经越过线的车辆的集合

            crossed\_line\_count = 0

            crossed\_vehicles = set()

            # 循环视频

            while cap.isOpened():

                # 读取一帧

                success, frame = cap.read()

                if success:

                    # 用户没有指定类别

                    if class4 == -1:

                        # Run YOLOv8 tracking 模型

                        results = model.track(frame, persist=True, conf=conf4, iou=IOU4, show=False)

                    # 用户指定了类别

                    else:

                        # Run YOLOv8 tracking 模型

                        results = model.track(frame, persist=True, conf=conf4, iou=IOU4, classes=class4, show=False)

                    # 拿到 boxes这个矩形框 和 track跟踪的IDs

                    boxes = results[0].boxes.xywh.cpu()

                    if results[0].boxes.id is not None:

                        track\_ids = results[0].boxes.id.int().cpu().tolist()

                        # 绘制结果到框frame中

                        annotated\_frame = results[0].plot()

                        # 根据不同车不同锚框来绘制轨迹

                        for box, track\_id in zip(boxes, track\_ids):

                            x, y, w, h = box

                            track = track\_history[track\_id]  # 不同车对应不同track

                            track.append((float(x), float(y)))  # x, y 锚框中心点坐标

                            # 超过100帧前的轨迹段会消失

                            if len(track) > 100:

                                track.pop(0)  # 弹出队列最前面的元素

                            # 绘制交通轨迹

                            points = np.hstack(track).astype(np.int32).reshape((-1, 1, 2))

                            cv2.polylines(annotated\_frame, [points], isClosed=False, color=(250, 170, 0), thickness=8)

                            # 检查车辆的中心点是否超过了越线的位置，并且车辆没有被记录过

                            if y < line\_position and track\_id not in crossed\_vehicles:

                                print(frame.shape[0] - line\_position)

                                crossed\_line\_count += 1

                                crossed\_vehicles.add(track\_id)

                            # 查看crossed\_line\_count的状态

                            print(crossed\_line\_count)

                            # 绘制越线检测线

                            cv2.line(annotated\_frame, (0, line\_position),

                                     (frame.shape[1], line\_position), (0, 255, 0),

                                     2)

                            # 在帧上绘制越线计数

                            cv2.putText(annotated\_frame, f'Crossed line count: {crossed\_line\_count}',

                                        (50, 100), cv2.FONT\_HERSHEY\_SIMPLEX, 2,

                                        (255, 200, 0), 3, cv2.LINE\_AA)

                        # 查看annotated\_frame格式

                        print(annotated\_frame)

                        # 将带有跟踪结果的帧添加到帧列表中

                        video\_frames.append(annotated\_frame)

                else:

                    # Break the 循环 if video 到达末尾

                    break

            # 获取视频帧的维度

            frame\_width = int(cap.get(3))

            frame\_height = int(cap.get(4))

            # 创建VideoWriter对象

            fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

            out = cv2.VideoWriter("video\_output/" + file\_name, fourcc, 20.0,

                                  (frame\_width, frame\_height))

            # 遍历带轨迹的帧列表，合成一起组成视频并写入到目标视频中

            for frame in video\_frames:

                # Write the annotated frame to the output video

                out.write(frame)

            # 输出越线车辆的数量

            print("Crossed line count:", crossed\_line\_count)

            # 释放摄像头流

            cap.release()

            # 关闭输出视频流

            out.release()

        except Exception as e:

            print(e)

        # 处理好后，展示轨迹视频

        try:

            self.media\_player4.setMedia(QMediaContent(QUrl("./video\_output/" + file\_name)))

            self.media\_player4.play()

            print("看看预测好的file\_path" + "./video\_output/" + file\_name)

        except Exception as e:

            print(e)

    # --- tab5 dataset ---

    def selectDataset(self):

        # 打开文件对话框，选择要读取的文件

        file\_path, \_ = QFileDialog.getOpenFileName(None, '选择要读取的文件', '.',

                                                   'Excel Files (\*.xlsx \*.xls);;CSV Files (\*.csv)')

        if file\_path:

            if file\_path.endswith('.xlsx'):

                print("正在读取 Excel 文件，请稍候...")

                try:

                    xlsx\_file = pd.ExcelFile(file\_path)

                    self.df = pd.read\_excel(xlsx\_file)

                except Exception as e:

                    print(e)

                print("读取成功！")

            elif file\_path.endswith('.csv'):

                # 读取 csv 文件

                print("正在读取 CSV 文件，请稍候...")

                self.df = pd.read\_csv(file\_path)

                print("读取成功！")

            try:

                # 设置表格的行数和列数并显示

                self.xlsx.setRowCount(self.df.shape[0])

                self.xlsx.setColumnCount(self.df.shape[1])

                self.show\_row.setText(str(self.df.shape[0]))

                self.show\_col.setText(str(self.df.shape[1]))

                # 设置列标签

                self.xlsx.setHorizontalHeaderLabels(self.df.columns.tolist())

                # 填充表格

                for i in range(self.df.shape[0]):

                    for j in range(self.df.shape[1]):

                        item = QTableWidgetItem(str(self.df.iloc[i, j]))

                        self.xlsx.setItem(i, j, item)

            except Exception as e:

                print("请输入有效的数据集！", e)

        else:

            print('用户取消了数据集选择操作')

            return

    def drawChart5(self):

        # 拿到用户输入id，如果没输入就MessageBox提示

        if self.input\_id.text() == '':

            print("请输入想要绘制轨迹的物体id")

            dialog = QMessageBox()

            dialog.setWindowTitle('warning')

            dialog.setIcon(QMessageBox.Warning)

            dialog.setText('请先输入想要绘制轨迹的物体id')

            dialog.exec\_()

            return

        # 输入后还要判断输入是否合法

        try:

            myinput = int(self.input\_id.text())

        except ValueError:

            # 如果用户输入的不是整数，显示一个错误提示框

            QMessageBox.warning(self, 'error', '请输入有效的整数', QMessageBox.Ok)

            return

        print(self.df)

        print(myinput)

        # 查询所有的行数据，根据id筛选出符合条件的行

        try:

            print(self.df['Track ID'] == myinput)

            data = self.df[self.df['Track ID'] == myinput]

            data = data.reset\_index(drop=True)

        except Exception as e:

            print(e)

        # 选择x,y两列数据绘制散点图

        print("正在画了...")

        # 位置图

        try:

            self.figure5 = Figure(figsize=(4, 3))

            self.myax = self.figure5.add\_subplot(111)

            self.canvas = FigureCanvas(self.figure5)

            x = data['X']

            y = data['Y']

            print(x)

            self.myax.set\_title('location x and y')

            self.myax.scatter(x, y, c='green')

            self.canvas.draw()

            scene = QGraphicsScene(self)

            scene.addWidget(self.canvas)

            self.chart5.setScene(scene)

            # 再把类别图也显示出来

            scene = QGraphicsScene()

            print(data['Category'])

            img = QPixmap(f"image/{data['Category'][0]}.png")

            img = img.scaled(200, 200)

            scene.addPixmap(img)

            self.show\_image5.setScene(scene)

        except Exception as e:

            print(e)

    # 导出位置图表

    def export\_chart5(self):

        # 拿到想要导出的路径

        file\_path, \_ = QFileDialog.getSaveFileName(self, 'Save Chart Image', '', 'Images (\*.png \*.jpg \*.bmp)')

        if file\_path:

            # 保存为图片

            self.figure5.savefig(file\_path)

            # 提示用户

            try:

                dialog = QMessageBox()

                dialog.setWindowTitle('success')

                dialog.setIcon(QMessageBox.Information)

                dialog.setText('导出成功！')

                dialog.exec\_()

            except Exception as e:

                print(e)

        else:

            print('用户取消了保存图片操作')

            return

    # --- tab 6 ---

    # 保存热力图

    def export\_chart6\_1(self):

        # 保存热力图

        pixmap = QPixmap()

        # 获取图表的截图

        pixmap = self.hotmap.grab()

        # 拿到想要导出的路径

        file\_path, \_ = QFileDialog.getSaveFileName(self, 'Save Chart Image', '', 'Images (\*.png \*.jpg \*.bmp)')

        if file\_path:

            # 保存为图片

            pixmap.save(file\_path)  # 保存图表

            # 提示用户

            try:

                dialog = QMessageBox()

                dialog.setWindowTitle('success')

                dialog.setIcon(QMessageBox.Information)

                dialog.setText('导出成功！')

                dialog.exec\_()

            except Exception as e:

                print(e)

        else:

            print('用户取消了保存图片操作')

            return

    # 保存类别词频图

    def export\_chart6\_2(self):

        # 拿到想要导出的路径

        file\_path, \_ = QFileDialog.getSaveFileName(self, 'Save Chart Image', '', 'Images (\*.png \*.jpg \*.bmp)')

        if file\_path:

            # 保存为图片

            self.figure6\_2.savefig(file\_path)

            # 提示用户

            try:

                dialog = QMessageBox()

                dialog.setWindowTitle('success')

                dialog.setIcon(QMessageBox.Information)

                dialog.setText('导出成功！')

                dialog.exec\_()

            except Exception as e:

                print(e)

        else:

            print('用户取消了保存图片操作')

            return

    # 退出函数

    def myexit(self):

        exit()