國立空中大學

「Python 程式設計與實務應用」

期末小組專題

「房價預測器」

應用程式開發手冊

導師：賴素純

班級：ZZZ002

組別：第 11 組

組名：8089

組長：112222911 詹秉蒼

組員（按 tronclass 平台組員順序）:

112224868 蔡惠婷

112224871 蔡婷羽

113122776 張心齡

113122877 陳銘泓

113170973 余誼姍

目　錄

一、 專題主題

房價預測器

二、 專題簡介

本專案開發之軟體，透過爬取內政部實價登錄資料，使用最真實的成交資料作為依據，以統計學方法科學地預測房價行情。

在現今臺灣房價飛漲，房價所得比不斷上升的年代，購屋已不易，又若購屋前沒有做足功課及進行相關統計分析，不熟悉欲購置區域的行情，就容易受騙，以高於行情的價格成交。

購屋前的資料蒐集及統計分析費時又費力，而透過本軟體，購屋者可藉由篩選特定條件的房屋交易資料，並輸入欲購置的時間及物件坪數等相關資料，就可以快速得到欲購買標的之預測房價。有了房屋預測價格，購屋時便不易遭屋主或房仲之話術欺騙，導致買貴。

本軟體可減輕購屋者購屋前資料收集分析的負擔，增加決策正確性及速度，並提供議價依據（議價籌碼）。

三、 小組分工

工作項目		人員
提案發想		蔡婷羽（112224871）
需求規格化		蔡婷羽（112224871）
專案統籌規劃		詹秉蒼（112222911）
需求分析		全體組員
程式撰寫	GUI 介面、數據驗證	陳銘泓（113122877）
	資料表設計與優化	詹秉蒼（112222911）
	交易資料下載	詹秉蒼（112222911）
	交易資料檔彙整與分類	余誼姍（113170973） 張心齡（113122776）
	資料庫查詢語句與條件產出	蔡惠婷（112224868）
	預測模型與演算法	蔡婷羽（112224871）
	整合程式碼	詹秉蒼（112222911）
應用程式開發手冊編纂		詹秉蒼（112222911） 蔡婷羽（112224871） 陳銘泓（113122877） 蔡惠婷（112224868）
使用手冊製作		余誼姍（113170973）
執行檔編譯（Windows）		陳銘泓（113122877）
執行檔編譯（MacOS）		蔡婷羽（112224871）
整合測試		全體組員
報告投影片製作		張心齡（113122776）
專題報告		張心齡（113122776）
宣傳影片/海報製作		余誼姍（113170973）

模組開發者	模組名稱
陳銘泓（113122877）	root.GUI.py
蔡婷羽（112224871）	root.predictive_model.py
蔡惠婷（112224868）	root.Select.py
余誼姍（113170973）	root.lib.DataFormatting.py（主程式未使用） root.output.*（主程式未使用）
詹秉蒼（112222911）	root.lib.Api.py root.lib.DownloadFile.py root.lib.CrateLvrData.py root.lib.MySQL.py root.lib.Tools.py root.lib.params.py
張心齡（113122776）	root.rearrange.py（主程式未使用）

四、 開發環境

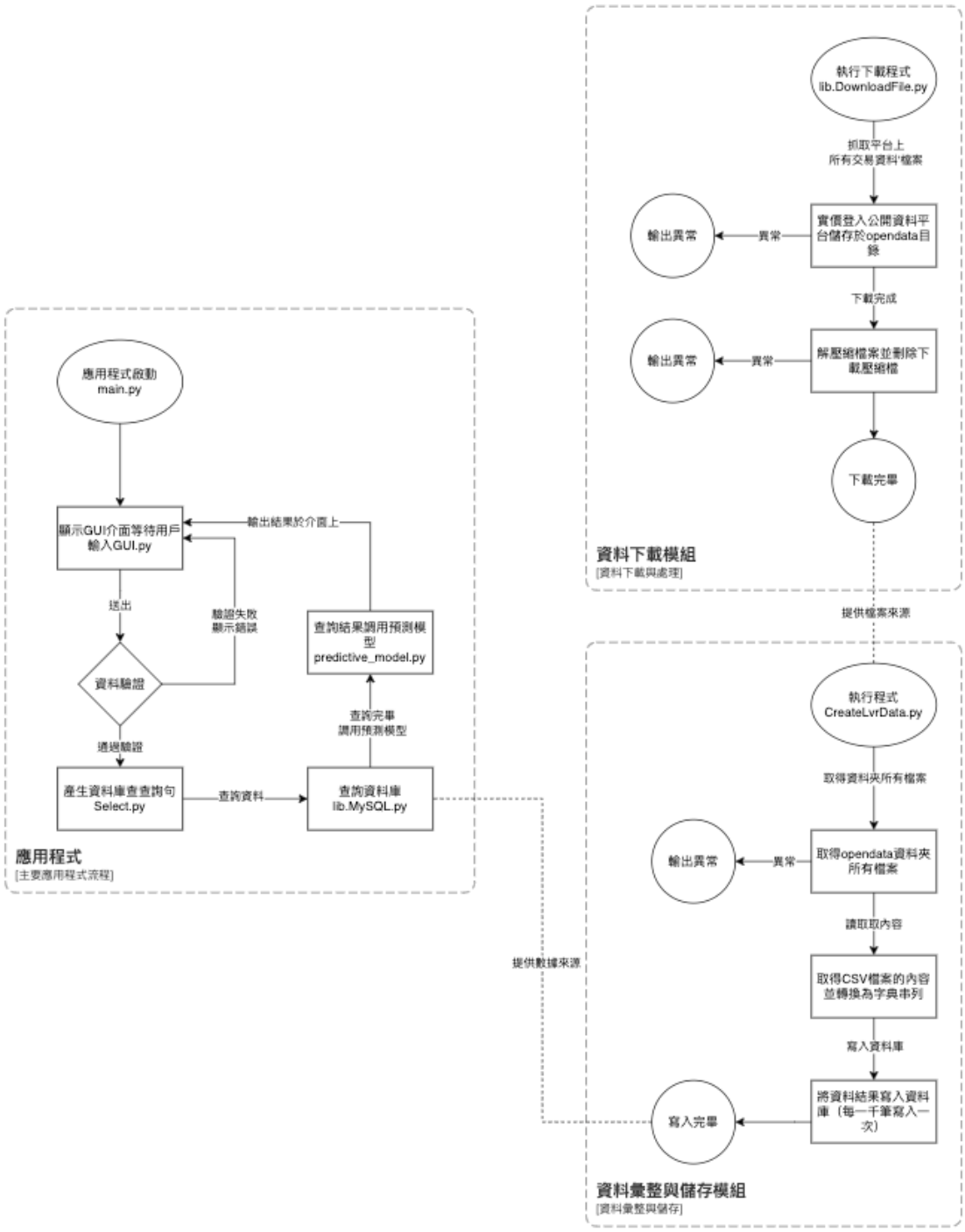
- 語言：Python 3.12.7
- 依賴套件：
  - 檔案下載：requests
  - 檔案解壓縮：zipfile
  - CSV 檔案處理：os、sys、csv、datetime、collections、json
  - 資料庫操作：mysql.connector
  - GUI 介面與資料驗證：tkinter、customtkinter、functools. Partial
  - 預測模型：pandas、numpy
- 開發系統：Windows、MacOS
- 流程圖：Draw.io
- 資料庫：MySQL 8.1.30
- 版本控制：Git、GitHub
- IDE：Microsoft Visual Studio Code

五、 系統開發流程

（一）需求分析

由全體組員進行討論，對於提案內容進行流程圖繪製、功能分解、使用套件、分配開發人員、制定函數、類別接口與參數相關工作分派。

系統流程圖如下：



（二）定義功能的輸出、輸入與邏輯規則

資料下載邏輯：

```
open data 下載規則與邏輯

最近資料下載位置 https://plvr.land.moi.gov.tw//Download?type=zip&fileName=lvr_landcsv.zip

前一期資料 下載下來的資料會有csv txt xml檔案同時存在
ht                                    v.tw/DownloadHistory?type=history&fileName=20241001
ht                                    v.tw/DownloadHistory?type=history&fileName=20241011
https://plvr.land.moi.gov.tw/DownloadHistory?type=history&fileName=20241021

season 季度資料
規則由101S1開始到113S4
https://plvr.land.moi.gov.tw//DownloadSeason?season=113S3&type=zip&fileName=lvr_landcsv.zip

檔案規則
範例 a_lvr_land_a_build.csv
第一區塊：檔案開頭的英文字代表區域 a 表示台北對照表請見lib/params.py中的city字典
第二區塊：lvr 固定字串
第三區塊：land固定字串
第四區塊：a b都是買賣資料 c為租賃資料，目前還不清楚ab的差異
第五區塊：land表示土地 build表示建物 park 表示車位
```

追蹤連結 (cmd + 按一下)

CSV 資料轉換為資料庫格式與注意事項：

資料表與結構

csv檔案中的欄位名稱	資料庫欄位名稱	資料型別	預設值	備註
	city_code	string		縣市代號 （參照params.py的city字典Key）
	city_name	string		縣市名稱 （參照params.py的city字典Key）
	town_code	string		鄉鎮市區參照params.py的town字典key）
鄉鎮市區	town_name	string		鄉鎮市區參照params.py的town字典value
交易標地	trade_sign	int		（房地 => 1 建物 => 2 土地=> 3 車位=>4, 房地+車位 => 5）
土地位置建物門牌	address	strnig		
交易年月日	trade_date	int		
成交總價(元)	price_total	int		
單價元平方公尺	price_nuit	int		
建物移轉總面積平方公尺	total_area	float		
編號	code	string		
屋齡	age	int	0	此處比較特特別， 假測現在開啟的檔案是a_lvr_land_a.csv要取這份檔案中的「編號」欄位的值，然後到a_lvr_land_a_build.csv這份檔案對應的「編號」取得屋齡，如果是土地就沒有屋齡問題預設值為0

GUI 介面提供的輸出內容：

```
# I/O 、 處理函式 及 變數部分 ==========================================================================================
# [I/O]輸入資料表
user_input_list = {
    "city":"",                  # str :     縣市,                                       必填：是 (預設:第一筆的key)
    "town":"",                  # str :     鄉鎮市區,                                   必填：是 (預設:第一筆的key)
    "ptype":[],                 # list :    1房地、2建物、3土地、4車位、5房地+車位,     必填：是 (預設:[1])
    "p_build": None,            # str :     門牌地址 :                                  必填：否 (預設:)
    "p_startY": None,           # int :     交易期間 (年起) 101-113,                    必填：是 (預設:101)
    "p_startM": None,           # int :     交易期間 (月起) 1-12,                       必填：是 (預設:1)
    "p_endY": None,             # int :     交易期間 (年迄) 101-113,                    必填：是 (預設:113)
    "p_endM": None,             # int :     交易期間 (月迄) 1-12,                       必填：是 (預設:12)
    "pmoney_unit": None,        # int :     單位 (1 => 萬元 , 2 => 元)                  必填：是 (預設:1)
    "minp": None,               # int :     最小值(單價),                               必填：否 (預設:)
    "maxp": None,               # int :     最大值 (單價) ,                             必填：否 (預設:)
    "unit": None,               # int :     面積單位 (1 => M^2 ，2 => 坪) ,             必填：是 (預設:2)
    "mins": None,               # int :     最小值 (坪數) ,                             必填：否 (預設:)
    "maxs": None,               # int :     最大值 (坪數) ,                             必填：否 (預設:)
    "avg_var": None,            # int :     屋齡,                                       必填：否 (預設:)

    # 計算部分 (我們自訂的)
    "calculate_Y": None,        # int :     目標期間 (年) ,                     必填：是 (預設:)
    "calculate_M": None,        # int :     目標期間 (月) ,                     必填：是 (預設:)
    "calculate_unit": None,     # int :     面積單位 (1 => M^2 ，2 => 坪) ,     必填：是 (預設:2)
    "calculate_area": None      # int :     面積,                               必填：是 (預設:)
}
```

預測模型的輸入與輸出：

```
    """
    本函式可回傳依據使用者輸入條件，所模擬計算出的預測房價單價 (萬元/坪)

    參數:
        user_input_list (dict): 從GUI介面取得的使用者輸入條件清單
        origin_data (dict): 交易年月及各該年月平均單價 (元/平方公尺) (資料按交易時間由舊到新排序)

    回傳:
        float: 預測房價單價 (萬元/坪)

    範例:
        >>> input_io_call(user_input_list)
        42.34489458593352
    """
```

（三）開發項目分工與制定完成日期

於每次會議討論工作項目、分配工作並追蹤進度，會議紀錄如下表：

會議時間	討論內容
113 年 11 月 5 日 19：00～20：00	1. 流程圖解說 2. 討論分工 3. 討論資料型態 4. 相關知識教學（網際網路、requests、API 等概念） 5. 初步擬定作業時程
113 年 11 月 19 日 21：00～22：20	1. 運行一遍程式的運作結果（由組長看完後合併到 develop 分支，由組長統一展示） 2. 各程式間的串接輸出與輸入的分析（所有屬性是否都存在、型別是否正常） 3. 指派程式串接事務，以及完成時間（１人，串接有問題，原開發者協助除錯與調整） 4. 著手進行文件、報告相關事宜（共同討論如何分工） 5. 作業時程規劃
113 年 12 月 3 日 19：00～19：35	1. 未盡事宜工作分配（開發手冊統整及使用手冊撰寫事宜） 2. 報告投影片展示及說明
113 年 12 月 13 日 21：00～	

（四）時程圖（甘特圖）

時間 / 任務名稱	1	2	3	4	5	6	7	8	9	10	11	12
題目發想、提案報告撰寫、需求分析												
需求規格化、程式碼撰寫、開發												
測試（單元測試、整合測試、跨平台測試）												
報告投影片製作												
面授報告												
開發文件、使用手冊、宣傳影片或海報製作												

註：1 表示專案進行第一週（10/13～10/19）；2 表示專案進行第二週（10/20～10/26）；以下以此類推。

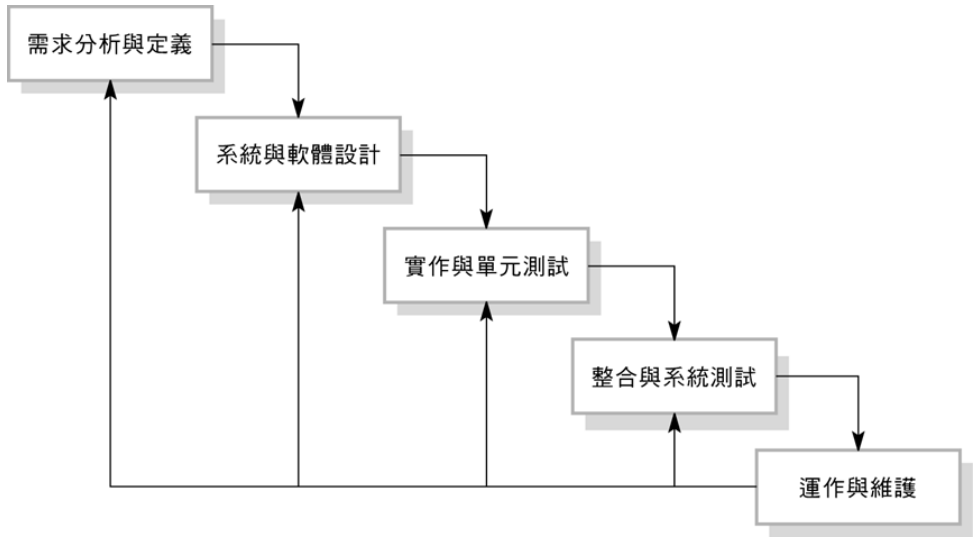
（五）軟體開發程序

軟體開發的程序有四種：瀑布式、漸進式、需求規格化、組合式的軟體開發程序。

本專案採傳統階梯式的軟體開發程序，亦即瀑布式的軟體開發程序，下圖為階梯式的軟體開發模型，也就是 Waterfall model。

此軟體開發程序，講求軟體生命週期（Software life cycle），將軟體發展劃分為明確階段。此開發模式為標準的程序，為大多數人所接受，有利於軟體專案的管理。

由前揭甘特圖可見得，本專案開發程序中，有明確的階段。在經過系統需求分析後，進行需求規格化，接著進行系統開發實作，實作的過程中，亦同時進行單元測試，實作完畢後，進行整合測試，故屬瀑布式的軟體開發程序。

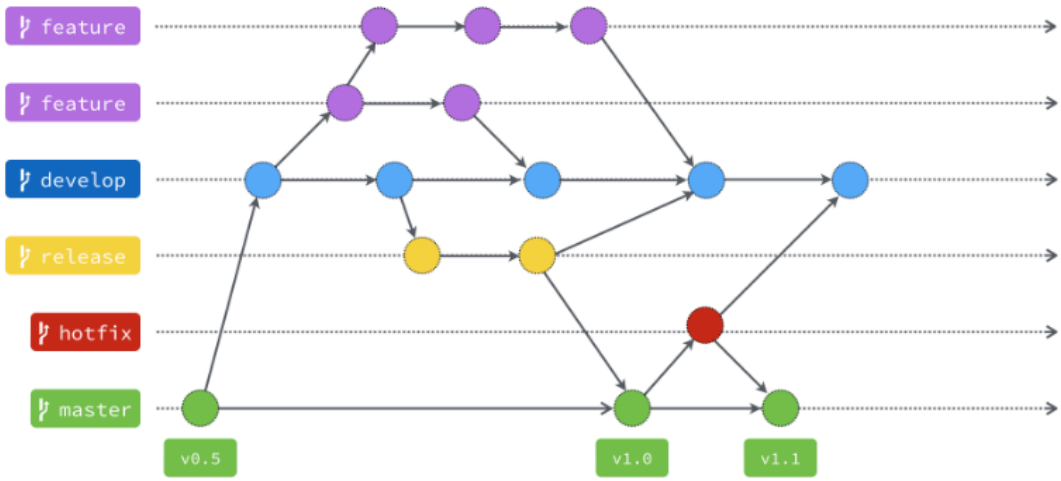


(圖片來源：https://wayne265265.pixnet.net/blog/post/113080214)

（六）版本控制協作流程

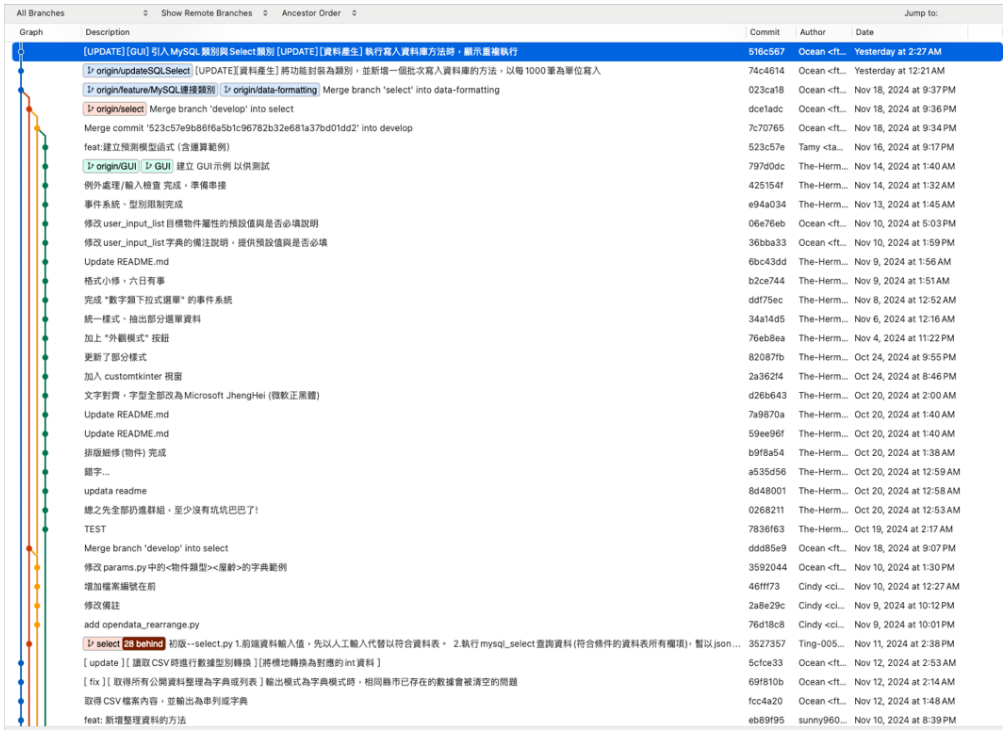
採用現今最流行的 GitFlow 流程，將 Branch（分支）狀態系統化。

組員開發各自所負責之功能時，於各自之 feature 分支進行開發，功能開發完畢後，合併至 develop 分支，待功能測試完畢，整體程式可正常執行，即將 develop 分支合併至 main 分支（master 分支），即成為穩定上線版本。



（圖片來源：https://gitbook.tw/chapters/gitflow/why-need-git-flow）

下圖為本專案版本控制協作流程歷程節錄：



（七）開發挑戰與問題解決

問題一：實價登入網站的版本更新，無法獲取資料來源。

解決方式：使用爬蟲抓取「不動產成交案件實際資訊資料供應系統」中的公開資料，來解決無法即時獲取交易資料問題，作為解決方案。

問題二：下載之數據量高達 460 萬餘筆，使用資料庫儲存時，遭遇效能問題。

解決方式：透過資料庫的分析語句工具對相關的查詢語句，進行索引的優化與調教。

（八）整合測試與問題修正

由組員各自在本機上進行完整的測試，組員使用之作業系統包含 Windows 及 macOS，故測試包含跨平台測試。

測試階段遇到問題反映給原作者，由原作者完成修正後，再進行測試。

六、 程式設計主要技巧

（一）導入模組

導入 Python 原生、第三方模組加速建構，以及組員開發之各實作程式，使開發工作可分工各自進行，不互相干擾。

```
1
2   # libraries Import
3   from functools import partial
4   from tkinter import *
5   import customtkinter
6   from lib.MySQL import MySQL
7   from Select import Select
8   from lib.Tools import Tools
9   from predictive_model import predictive_model
```

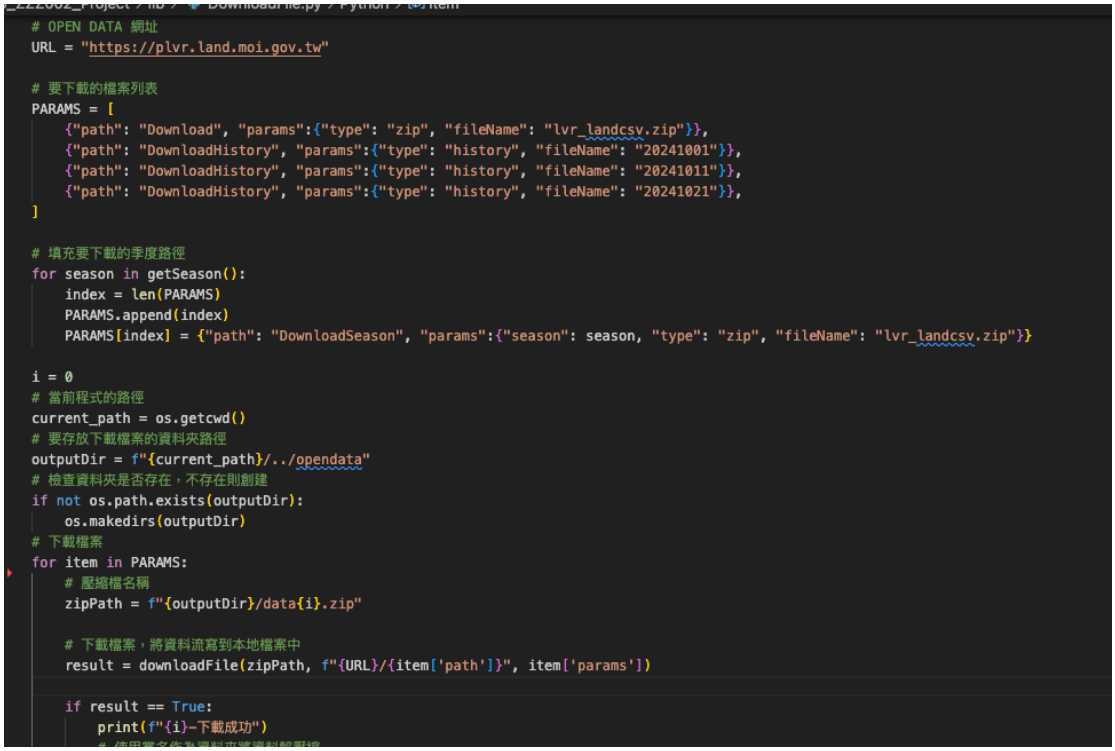
（二）類別方法封裝

下圖將資料庫的新增（C）、查詢（R）、刪除（D）、更新（U）各方法封裝為類別：

```
import mysql.connector
from mysql.connector import Error

class MySQL:
    '''
    提供MySQL的封裝類別
    在初始化時自動連接、並在結束時自動釋放連線
    提供select與delete與insert封裝方法，確保預防資料庫注入攻擊
    '''

    def __init__(self):…

    def query(self, sql, params=None):…

    def delete(self, sql, params=None):…

    def insert(self, sql, params=None):…

    def insert_many(self, sql, params_list):…

    def __enter__(self):…

    def __exit__(self, exc_type, exc_value, traceback):…
```

（三）HTTP 訪問

使用 requests 套件，下載「不動產成交案件實際資訊資料供應系統」中的交易資料。下圖為先產生所有欲下載的連結，透過 http GET 方法將所有公開檔案下載到本地端。



```python
# OPEN DATA 網址
URL = "https://plvr.land.moi.gov.tw"

# 要下載的檔案列表
PARAMS = [
    {"path": "Download", "params":{"type": "zip", "fileName": "lvr_landcsv.zip"}},
    {"path": "DownloadHistory", "params":{"type": "history", "fileName": "20241001"}},
    {"path": "DownloadHistory", "params":{"type": "history", "fileName": "20241011"}},
    {"path": "DownloadHistory", "params":{"type": "history", "fileName": "20241021"}},
]

# 填充要下載的季度路徑
for season in getSeason():
    index = len(PARAMS)
    PARAMS.append(index)
    PARAMS[index] = {"path": "DownloadSeason", "params":{"season": season, "type": "zip", "fileName": "lvr_landcsv.zip"}}

i = 0
# 當前程式的路徑
current_path = os.getcwd()
# 要存放下載檔案的資料夾路徑
outputDir = f"{current_path}/../opendata"
# 檢查資料夾是否存在，不存在則創建
if not os.path.exists(outputDir):
    os.makedirs(outputDir)
# 下載檔案
for item in PARAMS:
    # 壓縮檔名稱
    zipPath = f"{outputDir}/data{i}.zip"

    # 下載檔案，將資料流寫到本地檔案中
    result = downloadFile(zipPath, f"{URL}/{item['path']}", item['params'])

    if result == True:
        print(f"{i}-下載成功")
```

（四）檔案解壓縮與解析

將下載的檔案（壓縮檔進行解壓縮）完成後刪除，避免佔用過多空間。

```python
    # 下載檔案
for item in PARAMS:
    # 壓縮檔名稱
    zipPath = f"{outputDir}/data{i}.zip"

    # 下載檔案，將資料流寫到本地檔案中
    result = downloadFile(zipPath, f"{URL}/{item['path']}", item['params'])

    if result == True:
        print(f"{i}-下載成功")
        # 使用黨名作為資料夾將資料解壓縮
        unzipResult = UnZip(zipPath, zipPath.replace(".zip", ""))
        if unzipResult :
            print(f"{i}-解壓縮完成")
            if os.path.exists(zipPath):
                os.remove(zipPath)
                print(f"{i}-已刪除下載的ZIP檔")
        else:
            print(f"{i}-解壓縮失敗")
    else:
        print(f"{i}-下載失敗")
    i += 1
```

使用 CSV 套件以取得檔案內容，並彙整成資料庫所需格式。

```python
def getData(self, outputModel = 1, rows = -1):
    '''
    從opendata資料夾中取得對應的csv檔數據，
    發現a_lvr_land_b.csv後面為_b的檔案類型，因該是土地+車位，因該也可以略過不抓
    Args:
        outputModel: (int) 0 => 輸出字典模式 e.g {A:[],B:[]} 1 => 串列模式 e.g. [{item1...},{item2...}]
    '''
    # 取當前目錄
    currentDir = os.path.dirname(os.path.abspath(__file__))
    # opendata目錄
    openddataDir = os.path.join(currentDir, '..', 'opendata')
    # 取所有子目錄列表（這種寫的問題是在萬一參雜了非需要的資料目錄可能會在讀取csv時出錯）
    dirList = [os.path.join(openddataDir, d) for d in os.listdir(openddataDir) if os.path.isdir(os.path.join(openddataDir, d))]
    # 儲存最終結果的列表
    if outputModel == 1:
        result = []
    else:
        result = {}
    # 迴圈所有的data資料夾
    for index, dirPath in enumerate(dirList):
        if index == rows:
            break
        # 處理檔案中的a-z開頭
        for fi, i in enumerate(range(ord('a'), ord('z') + 1)):
            if fi == rows:
                break
            key = chr(i)
            # 如果需要抓其他後綴檔案 像是xxx_b.csv _c.csv，可以在多一曾回圈先處理檔案命稱
            # for prefix in ["a", "b", "c"]:
            #   fileList = [f"{key}_lvr_land_{prefix}_{suffix}.csv" for suffix in ['build', 'land', 'park']]
            #   fileList.insert(0, f"{key}_lvr_land_{prefix}.csv")
            # ... 後面的邏輯就都差不多
            try:
                # 後來對照比對後，發現_b的檔案是土地＋車位資料，也可略過不抓
                prefix = os.path.join(dirPath, f"{key}_lvr_land_a")
                fileList = [f"{prefix}_{suffix}.csv" for suffix in ['build', 'land', 'park']]
                fileList.insert(0, f"{prefix}.csv")
                # 取檔案內容
                print(f"處理檔案：{fileList[0]}")
```

（五）資料庫批次寫入

因搜集到的數據量超過百萬筆（約 460 萬餘筆），若一次性寫入資料庫，將造成大量數據庫開銷，故採用每一千筆批次寫入，使資料庫的 IO 達到較良好的效能。

```python
def insertSQL(self, row = 1000):
    '''
    將數據寫入資料庫每1000比進行一次批次寫入
    '''
    result = self.getData(1)
    total = len(result)
    print(f"共有{total // row + 1}批資料要寫入每批資料有{row}筆")
    with MySQL.MySQL() as db:
        sql = []
        sql.append("INSERT INTO omeiliau_nou.lvr_lnd (city_code, city_name, town_code, town_name, trade_sign, address, trade_date, price_total, price_n
        sql.append(f"VALUES ({(", ".join(["%s"] * 12)})")
        data_to_insert = []
        for idx,item in enumerate(result):
            #將字典轉元組
            data_to_insert.append((item["city_code"],item["city_name"],item["town_code"],item["town_name"],item["trade_sign"],item["address"],item["tra
            # db.insert_many(" ".join(sql), data_to_insert)
            if len(data_to_insert) >= row or idx == total - 1:
                db.insert_many(" ".join(sql), data_to_insert)
                data_to_insert = []  # 清空資料，為下一批次做準備
                print(f"已批次寫入資料：第 {idx // row + 1} 批")
```

（六）圖形化使用者介面（Graphical User Interface, GUI）

透過圖形化使用者介面搜集之用戶對資料的篩選條件，以此為基礎搜尋資料庫中之對應資料，並生成預測結果。

（七）數據驗證

提醒用戶必要輸入的數據，以及進行數據型態的控制與驗證。

七、 原始碼重點解說

（一）資料下載與解壓縮（lib.DownloadFile.py）

主要目的用於下載自 101 年起至當下之實價登錄資料，並將其解壓縮後儲存於本地端。主要邏輯片段：

函數主要產生 101 年到當年的年+季以[明國年] S [第幾季]為返回格式。

```python
def getSeason():
    '''
    取得從101年到當前的年+季
    Return:
        list 格式為 {年份}S{第幾季}
    '''
    # 取現在的西元年轉換為民國
    current_year = datetime.now().year
    ORGY = current_year - 1911
    #現在的季度
    current_quarter = ((datetime.now().month - 1) // 3) + 1
    items = []
    # 迴圈跑年
    for year in range(101, ORGY+1):
        # 迴圈跑季
        for s in range(1, 5):
            if ORGY == year and s >= current_quarter:
                break
            index = len(items)
            items.append(index)
            items[index] = f"{year}S{s}"
    return items
```

透過迴圈產生出對應公開資料平台的下載點 URL。

以字典串列形式返回，字典中包含檔名、路徑、類型。

```python
# OPEN DATA 網址
URL = "https://plvr.land.moi.gov.tw"

# 要下載的檔案列表
PARAMS = [
    {"path": "Download", "params":{"type": "zip", "fileName": "lvr_landcsv.zip"}},
    {"path": "DownloadHistory", "params":{"type": "history", "fileName": "20241001"}},
    {"path": "DownloadHistory", "params":{"type": "history", "fileName": "20241011"}},
    {"path": "DownloadHistory", "params":{"type": "history", "fileName": "20241021"}},
]

# 填充要下載的季度路徑
for season in getSeason():
    index = len(PARAMS)
    PARAMS.append(index)
    PARAMS[index] = {"path": "DownloadSeason", "params":{"season": season, "type": "zip", "fileName": "lvr_landcsv.zip"}}
```

將資料解壓縮後，再刪除壓縮檔以節省儲存空間。

```
for item in PARAMS:
    # 壓縮檔名稱
    zipPath = f"{outputDir}/data{i}.zip"

    # 下載檔案，將資料流寫到本地檔案中
    result = downloadFile(zipPath, f"{URL}/{item['path']}", item['params'])

    if result == True:
        print(f"{i}-下載成功")
        # 使用黨名作為資料夾將資料解壓縮
        unzipResult = UnZip(zipPath, zipPath.replace(".zip", ""))
        if unzipResult :
            print(f"{i}-解壓縮完成")
            if os.path.exists(zipPath):
                os.remove(zipPath)
                print(f"{i}-已刪除下載的ZIP檔")
        else:
            print(f"{i}-解壓縮失敗")
    else:
        print(f"{i}-下載失敗")
    i += 1
```

（二）資料庫連接與 CRUD 操作（lib.MySQL.py）

主要用於封裝 mysql.connector 套件中的資料操作方法，方便重複調用，以及於每次操作資料庫完畢後，自動釋放連線資源。

封裝 CRUD Create 、Read、Update、Delete 資料操作方法，並再提供一個針對批次寫入的方法。

```
    def query(self, sql, params=None):
        """
        執行查詢並返回結果
        with MySQL() as db:
            result = db.query("SELECT * FROM lvr_lnd WHERE column = %s", (value,))
        """
        results = []
        if self.connection:⋯
        return results

    def delete(self, sql, params=None):
        """
        執行刪除操作並提交變更
        with MySQL() as db:
            db.delete("DELETE FROM lvr_lnd WHERE column = %s", (value,))
        """
        if self.connection:⋯

    def insert(self, sql, params=None):
        """
        執行插入操作並提交變更
        with MySQL() as db:
            db.insert("INSERT INTO lvr_lnd (column1, column2) VALUES (%s, %s)", ("value1", "value2"))
        """
        if self.connection:⋯

    def insert_many(self, sql, params_list):
        """
        執行批次插入操作並提交變更
        with MySQL() as db:⋯
        """
        if self.connection:⋯
```

透過__enter__、__exit__在結束時自動釋放資料庫連線。

```
    def __enter__(self):
        """允許 with 語句使用此類別"""
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        """結束時自動關閉連線"""
        if self.connection and self.connection.is_connected():
            self.connection.close()
            print("資料庫連線已關閉")
```

（三）寫入資料庫（CreateLvrData.py）

主要用於提取所有下載的 CSV 檔案中的內容，並整理為可寫入資料庫的數據型態，將資料儲存於 MySQL 資料庫中。

CreateLvrData.getData()方法：
取得 lib.DownloadFile.py 下載的 CSV 檔案並從中提取數據，進行數據的資料轉換。

```
    # 迴圈所有的data資料夾
    for index, dirPath in enumerate(dirList):
        if index == rows:
            break
        # 處理檔案中的a-z開頭
        for fi, i in enumerate(range(ord('a'), ord('z') + 1)):
            if fi == rows:
                break
            key = chr(i)
            try:
                # 後來對照比對後，發現_b的檔案是土地＋車位資料，也可略過不抓
                prefix = os.path.join(dirPath, f"{key}_lvr_land_a")
                fileList = [f"{prefix}_{suffix}.csv" for suffix in ['build', 'land', 'park']]
                fileList.insert(0, f"{prefix}.csv")
                # 取檔案內容
                print(f"處理檔案：{fileList[0]}")
                main = self.getCsv(fileList[0], 'main', key) if os.path.exists(fileList[0]) else None
                build = self.getCsv(fileList[1], 'build', key) if os.path.exists(fileList[1]) else None

                # 沒有檔案的話就跳過
                if main == None:
                    continue
                # 把build轉換把code(編號)作為key
                build_dict = {item["code"]: item["age"] for item in build} if build != None else {}

                # 取的對應的屋齡
                for item in main:
                    code = build_dict.get(item['code'], 0)
                    item['age'] = code

                # 將資料加入到輸出結果
                if outputModel == 1:
                    result.extend(main)
                else:
                    if key.upper() in result:
                        result[key.upper()].extend(main)
                    else:
                        result[key.upper()] = main
            except ValueError as e:
                print(f"檔案：{fileList[0]}, 發生異常{e}")
            except Exception as e:
                print(f"檔案：{fileList[0]}, 發生異常{e}")
    return result
```

CreateLvrData.getCsv()方法：

對 CSV 中的數據進行預處理與轉換。

```
# 使用字典方式讀取csv
csv_reader = csv.DictReader(file)
# 要返回的列表
data = []
for i, d in enumerate(csv_reader):
    # 檔案第一行跳過（檔頭不需要）
    if i == 0:
        continue

    # 取得在map字典中有對應的key產生出新的字典
    result = {}
    for key, value in d.items():
        if(key in map):
            if map[key] == 'age':
                result[map[key]] = (int(value) if value.isdigit() else 0)
            elif map[key] == 'trade_sign':
                result[map[key]] = tradeMap.get(value, 0)
            elif map[key] == 'price_nuit':
                result[map[key]] = (int(value) if value.isdigit() else 0)
            else:
                result[map[key]] = value

    # 如果是主檔，添加縣市與鄉鎮市相關資訊
    if fileType == 'main':
        result = {
            'city_code': cityCode,
            'city_name': params.city[cityCode],
            'town_code': next((item["code"] for item in params.town[cityCode] if item["title"] == result["town_name"]), None)
            'age': 0
        } | result

    _k = len(data)
    data.append(_k)
    data[_k] = result
return data
```

CreateLvrData.insertSQL ()方法：

將取得的數據以每批一千筆的的速率寫入資料庫。

```
def insertSQL(self, row = 1000):
    '''
    將數據寫入資料庫每1000比進行一次批次寫入
    '''

    result = self.getData(1)
    total = len(result)
    print(f"共有{total // row + 1}批資料要寫入每批資料有{row}筆")
    with MySQL.MySQL() as db:
        sql = []
        sql.append("INSERT INTO omeiliau_nou.lvr_lnd (city_code, city_nam
        sql.append(f"VALUES ({(", ".join(["%s"] * 12))})")
        data_to_insert = []
        for idx,item in enumerate(result):
            #將字典轉元組
            data_to_insert.append((item["city_code"],item["city_name"],it
            # db.insert_many(" ".join(sql), data_to_insert)
            if len(data_to_insert) >= row or idx == total - 1:
                db.insert_many(" ".join(sql), data_to_insert)
                data_to_insert = []  # 清空資料，為下一批次做準備
                print(f"已批次寫入資料：第 {idx // row + 1} 批")
```

（四）GUI 介面（資料搜集）

使用 tkinter 及 customtkinter 套件製作 GUI 介面，允許用戶通過 GUI 介面圖像化輸入資料，介面中物件程式碼範本如下:

```
Class_1_optionMenu_1 = customtkinter.CTkOptionMenu(
    master=background_frame_1,
    values=list(city_dict.values()),
    command=update_town_options,
    font=("Microsoft JhengHei", 14),
    height=40,
    width=60,
    corner_radius=8,
)
Class_1_optionMenu_1.place(x=10, y=20)
```

使用元件包括: customtkinter 的 CTkScrollableFrame、CTkFrame、CTkButton、CTkLabel、CTkOptionMenu、CTkCheckBox、CTkRadioButton 及 CTkEntry。

使用者填寫資料時，會通過回調函數將資料寫入 user_input_list:

```
# 通用回調函數，根據選取值查詢對應字典，並匯入 user_input_list
def get_selected_key(selected_value, lookup_dict, code):
    global user_input_list
    for key, value in lookup_dict.items():
        if value == selected_value:
            user_input_list[code] = key
            print(f"[GUI]user_input_list.{code}: {user_input_list[code]}")
            return user_input_list[code]
```

（五）數據驗證

當使用者按下[生成資料]按鈕時，會開始檢查輸入資料是否正確、完整:

```
# 生成結果按鈕(一階判斷)
def on_output_button():
    print("\n[GUI]開始檢查輸入資料表:")
    can_output = True
    wrong_time_range = False
    warning_text = "警告: "

    # 縣市
    if user_input_list["city"] == "" or user_input_list["city"] == None :
        print("[GUI]警告! <city>項目為空! 該項目為必填!")
        can_output = False
        warning_text += "縣市、"
```

檢查函數分別驗證每項資料是否合規，若不符合規則，則判斷是否有預設值。若有，則填入預設值；若無，則對使用者進行文字警告:

```
# 輸出警告文字或調用計算函數
if(can_output == True):
    print("[GUI]輸入正確，調用計算函數")
    gui_output_float = input_io_call(user_input_list)

    if (user_input_list["calculate_area"] == None or user_input_list["calculate_area"] == 0):
    else:…

    Output_label.configure(text=output_text, text_color="#5555FF")

# 僅有起訖時間範圍錯誤
elif (warning_text == "警告: " and wrong_time_range == True):
    warning_text = "警告: 起訖時間範圍錯誤"
    Output_label.configure(text=warning_text, text_color="#CC0000")

# 輸入有缺漏
else:
    # 找到最後一個 "、" 並替換為 " 為必填項目!"
    warning_text = warning_text[::-1].replace("、", "!目項填必為 ", 1)[::-1]
    if (wrong_time_range == True):
        warning_text += "\n警告: 起訖時間範圍錯誤"
    Output_label.configure(text=warning_text, text_color="#CC0000")
```

如果資料填寫完整且合規，則呼叫資料庫及預測模型:

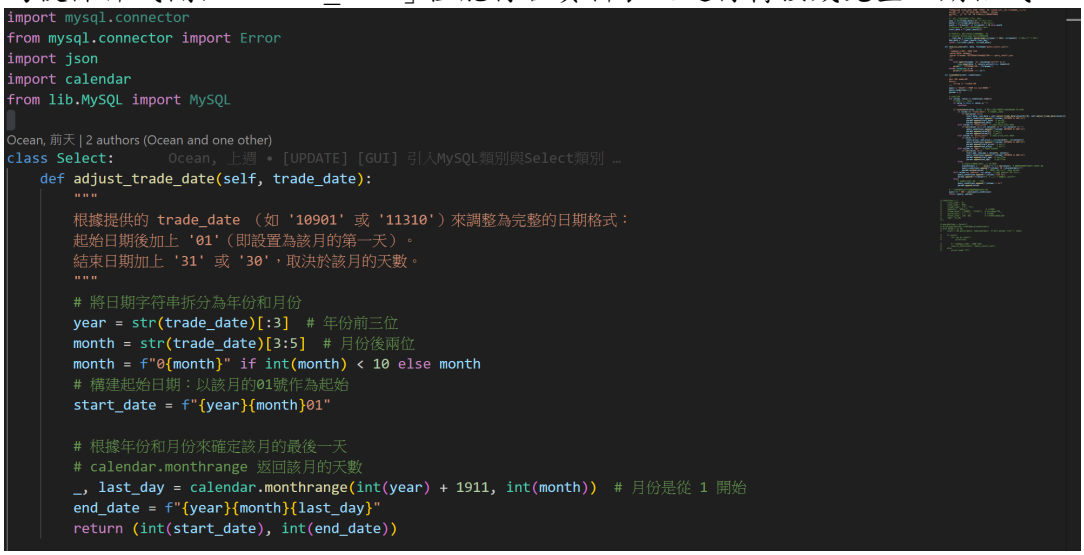
```python
# 輸入/輸出 串接函式 ================================================================
def input_io_call(dict):
    tools = Tools()
    sqlParams = tools.changeToSelectDict(dict)
    queryBuilder =  Select().createQuery(sqlParams)
    with MySQL() as db:
        print("資料庫查詢中・・・")
        result = db.query(queryBuilder[0], queryBuilder[1])
        data = tools.getKeyByDict(result)
        userInput = {
            "calculate_Y": dict['calculate_Y'],
            "calculate_M": dict['calculate_M'],
        }
        amount = predictive_model(userInput, data)
    print(dict)

    # 一律回傳 萬元/每單位面積。 須注意單位為何 "calculate_unit" （1 => M^2 ，2 => 坪）
    return round(amount, 2)
```

（六）資料庫語句生成（Select.py）

主要用於接收 GUI 介面所搜集到的用戶輸入資料，並產生對應的資料庫查詢語句

Select.createQuery()方法：
處理傳入的數據，產生對應的資料庫查詢語句，若數據沒有值或者為 None 時，將不會產生對應的欄位查詢語句。

定義函數 def adjust_trade_date()：
為使條件式欄位「trade_date」值能符合資料表而進行轉換成完整日期格式。



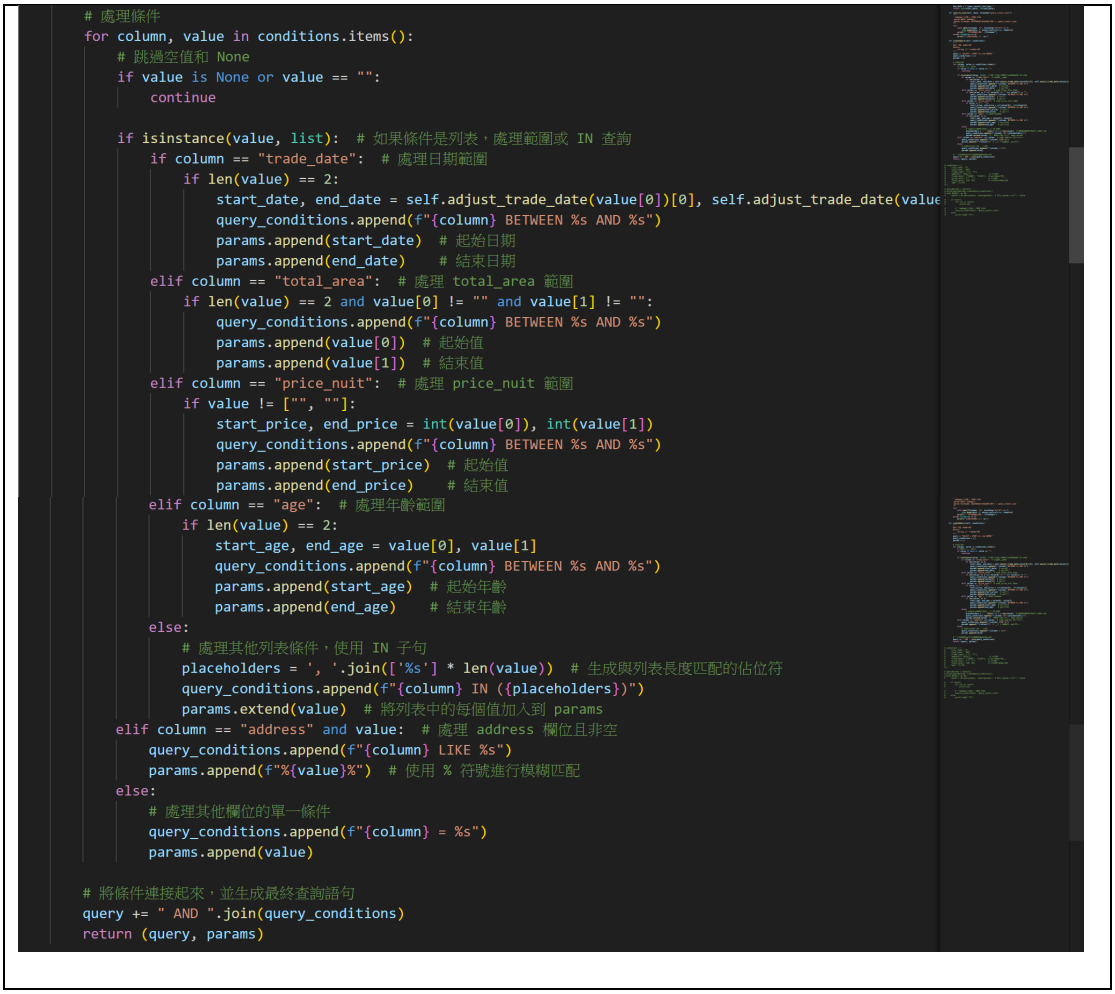
```python
import mysql.connector
from mysql.connector import Error
import json
import calendar
from lib.MySQL import MySQL

Ocean, 前天 | 2 authors (Ocean and one other)
class Select:       Ocean, 上週 • [UPDATE] [GUI] 引入MySQL類別與Select類別 …
    def adjust_trade_date(self, trade_date):
        """
        根據提供的 trade_date （如 '10901' 或 '11310'）來調整為完整的日期格式：
        起始日期後加上 '01'（即設置為該月的第一天）。
        結束日期加上 '31' 或 '30'，取決於該月的天數。
        """
        # 將日期字符串拆分為年份和月份
        year = str(trade_date)[:3]  # 年份前三位
        month = str(trade_date)[3:5]  # 月份後兩位
        month = f"0{month}" if int(month) < 10 else month
        # 構建起始日期：以該月的01號作為起始
        start_date = f"{year}{month}01"

        # 根據年份和月份來確定該月的最後一天
        # calendar.monthrange 返回該月的天數
        _, last_day = calendar.monthrange(int(year) + 1911, int(month))  # 月份是從 1 開始
        end_date = f"{year}{month}{last_day}"
        return (int(start_date), int(end_date))
```

定義函數 def save_to_json()：
將最後查詢資料存入 json 檔，用於預測模型。

```python
    def save_to_json(self, data, filename="query_result.json"):
        """
        將查詢結果保存為 JSON 檔案
        :param data: 查詢結果
        :param filename: 要儲存的檔案名稱，預設為 query_result.json
        """
        try:
            with open(filename, "w", encoding="utf-8") as f:
                json.dump(data, f, ensure_ascii=False, indent=4)
            print(f"資料已成功保存為 {filename}")
        except Exception as e:
            print(f"保存檔案時出錯: {e}")
```

定義函數 def createQuery()：
利用 if-elif-else 查詢資料表，先判斷以下欄位名稱後，再因應欄位屬性進行其它判斷。

```python
    def createQuery(self, conditions):
        """
        動態 SQL 查詢語句
        Return
            string 哩奧庫查詢語句
        """
        query = "SELECT * FROM lvr_lnd WHERE "
        query_conditions = []
        params = []
```



```
    # 處理條件
    for column, value in conditions.items():
        # 跳過空值和 None
        if value is None or value == "":
            continue

        if isinstance(value, list):  # 如果條件是列表，處理範圍或 IN 查詢
            if column == "trade_date":  # 處理日期範圍
                if len(value) == 2:
                    start_date, end_date = self.adjust_trade_date(value[0])[0], self.adjust_trade_date(value
                    query_conditions.append(f"{column} BETWEEN %s AND %s")
                    params.append(start_date)  # 起始日期
                    params.append(end_date)    # 結束日期
            elif column == "total_area":  # 處理 total_area 範圍
                if len(value) == 2 and value[0] != "" and value[1] != "":
                    query_conditions.append(f"{column} BETWEEN %s AND %s")
                    params.append(value[0])  # 起始值
                    params.append(value[1])  # 結束值
            elif column == "price_nuit":  # 處理 price_nuit 範圍
                if value != ["", ""]:
                    start_price, end_price = int(value[0]), int(value[1])
                    query_conditions.append(f"{column} BETWEEN %s AND %s")
                    params.append(start_price)  # 起始值
                    params.append(end_price)    # 結束值
            elif column == "age":  # 處理年齡範圍
                if len(value) == 2:
                    start_age, end_age = value[0], value[1]
                    query_conditions.append(f"{column} BETWEEN %s AND %s")
                    params.append(start_age)  # 起始年齡
                    params.append(end_age)    # 結束年齡
            else:
                # 處理其他列表條件，使用 IN 子句
                placeholders = ', '.join(['%s'] * len(value))  # 生成與列表長度匹配的佔位符
                query_conditions.append(f"{column} IN ({placeholders})")
                params.extend(value)  # 將列表中的每個值加入到 params
        elif column == "address" and value:  # 處理 address 欄位且非空
            query_conditions.append(f"{column} LIKE %s")
            params.append(f"%{value}%")  # 使用 % 符號進行模糊匹配
        else:
            # 處理其他欄位的單一條件
            query_conditions.append(f"{column} = %s")
            params.append(value)

    # 將條件連接起來，並生成最終查詢語句
    query += " AND ".join(query_conditions)
    return (query, params)
```

（七）預測模型（predictive_model.py）

用於接收透過資料庫查詢語句挑出的資料，並使用統計方法，運算出預測不動產單價後，將其回傳。

predictive_model.predictive_model()方法：
本方法有兩個參數，第一個參數為由 GUI 介面取得之使用者輸入條件清單；第二個參數為透過資料庫查詢語句挑出的資料（交易年月及各該年月平均單價（元/平方公尺）（資料按交易時間由舊到新排序））。
引入用於數據分析與處理之第三方函式庫 Pandas 及 NumPy 進行不動產成交價分析與預測。Pandas 為用於高階資料結構（如 DataFrame）和資料處理之工具；NumPy 用於數值計算，提供陣列操作和進階的數學函式。

```
import pandas as pd
import numpy as np
```

本方法將交易時間作為解釋變數（自變數），將不動產交易單價作為被解釋變數（依變數），透過統計學之「最小平方法」，求出簡單線性迴歸式，亦即不動產單價預測公式，再透過使用者輸入的欲購置時間，以產生對應該時點的預測不動產單價。

```python
# 預測函式
# 根據使用者輸入，計算迴歸公式，產出預測單價（萬元/坪），供GUI呼叫並計算（單價*目標面積（坪）=預測總價（萬元））以顯示預測結果（單價或總價，
# 如欲顯示總價，則需要再做額外運算）
def predictive_model(user_input_list, origin_data):
    """
    本函式可回傳依據使用者輸入條件，所模擬計算出的預測房價單價（萬元/坪）

    參數:
        user_input_list (dict): 從GUI介面取得的使用者輸入條件清單
        origin_data (dict): 交易年月及各該年月平均單價（元/平方公尺）（資料按交易時間由舊到新排序）

    回傳:
        float: 預測房價單價（萬元/坪）

    範例:
        >>> input_io_call(user_input_list)
        42.34489458593352
    """

    # 應傳入之參數origin_data（字典）範例：
    # {
    #     11110: 116730, # 111年10月之平均成交價為每平方公尺116730元
    #     11205: 121716,
    #     11307: 124736
    # }
    # key為「交易年月」，value為「各該月平均單價（元/平方公尺）」（資料按交易時間由舊到新排序）

    # 將交易資料字典的key（交易年月）提取出來，形成串列，存入變數trade_time_month
    trade_time_month = list(origin_data.keys())
    # 範例：trade_time_month = [11110, 11205, 11307]

    # 將交易資料字典的value（各交易年月平均單價（元/平方公尺））提取出來，形成串列，存入變數house_price_per_square_meter_month_average
    house_price_per_square_meter_month_average = list(origin_data.values())
    # 範例：house_price_per_square_meter_month_average = [116730, 121716, 124736]

    # 轉換變數trade_time_month的元素（交易年月）（最早的交易年月為0，其餘以此類推），儲存為變數trade_time
    # 將串列中第一個元素作為基期年月（因為資料庫在挑資料時已經按交易時間由舊到新排序，所以列表中第一個元素為交易時間最早的資料）
    base_year = trade_time_month[0] // 100 # 提取基期年份
    # 範例：base_year = 111（年）
    base_month = trade_time_month[0] % 100 # 提取基期月份
    # 範例：11110%100 = 10（月）
    # 範例：base_month = 10
    # 計算串列中每個元素與基期的月份差距
    trade_time = [] # 先創建交易年月轉換結果的空串列，等下再用迴圈把轉換後結果一一加進本串列
    for year_month in trade_time_month: # year_month指串列trade_time_month中每個單一元素（如：11110）
        year = year_month // 100 # 提取年份
        month = year_month % 100 # 提取月份
        # 差一年代表差12個月，計算年份差與月份差，以得到月份差距
        month_difference = (year - base_year) * 12 + (month - base_month)
        trade_time.append(month_difference)
    # 範例：trade_time = [0, 7, 21]
    # （111年10月與基期111年10月相差0個月；112年5月與111年10月相差7個月；113年7月與111年10月相差21個月）

    # 取得使用者輸入的欲購置交易年月，並儲存為變數buy_time_year與buy_time_month
    buy_time_year = user_input_list["calculate_Y"] # 從使用者輸入字典中提取key為calculate_Y所對應之value（目標期間（年））
    # 範例：buy_time_year = 114
    buy_time_month = user_input_list["calculate_M"] # 從使用者輸入字典中提取key為calculate_M所對應之value（目標期間（月））
    # 範例：buy_time_month = 3

    # 轉換變數buy_time_year與buy_time_month，並儲存為變數buy_time
    buy_time = (buy_time_year - base_year) * 12 + (buy_time_month - base_month)
    # 範例：buy_time = 29
    # （欲購置時間114年3月與基期111年10月相差29個月）

    # 產生預測公式（最小平方迴歸法擬合）
    data = pd.DataFrame({'X': trade_time, 'Y': house_price_per_square_meter_month_average})
    # pd.DataFrame(字典資料)
    # 範例：trade_time = [0, 7, 21]
    # 範例：house_price_per_square_meter_month_average = [116730, 121716, 124736]

    cofficients = np.polyfit(data['X'], data['Y'], 1)
    # 使用numpy的polyfit函式 進行 最小平方法擬合
    # 1 表示擬合 「一次」多項式模型（直線）
    # np.polyfit(字典[對應自變數的key], 字典[對應應變數的key], 想要擬合幾次多項式)
    # cofficients = [斜率, 截距]

    slope = cofficients[0] # 斜率
    intercept = cofficients[1] # 截距

    # 預測單價（元/平方公尺）公式
    predicted_house_price_per_square_meter = intercept + slope * buy_time
    # 轉換為單位之單位為「萬元/坪」
    predicted_house_price_per_pin = predicted_house_price_per_square_meter / 10000 / 0.3025

    # 回傳預測單價（萬元/坪）
    return predicted_house_price_per_pin
    # 範例：42.34489458593352（萬元/坪）
```

（八）結果輸出

output_show()函數取得回傳值後，判斷回傳值是計算結果(正數)或錯誤代碼(負數)，並以此輸出結果至 GUI:

```python
# GUI輸出顯示函式
def output_show(gui_output_float):
    global user_input_list

    # 形式正確的回傳值
    if(gui_output_float > 0):
        if (user_input_list["calculate_area"] == None or user_input_list["calculate_area"] == 0):
            # 輸入值無面積時，返回 萬元/坪(or 平方米)
            if(user_input_list["calculate_unit"] == 2):     # 單位:坪
                output_text = f"預期價格: {gui_output_float}萬元/坪"
            elif(user_input_list["calculate_unit"] == 1):   # 單位:平方米
                output_text = f"預期價格: {gui_output_float}萬元/平方米"
            else:
                print("[GUI]錯誤! 未定義的單位")
        else:
            # 輸入值有面積時，返回 總價-萬元
            output_text = f"預期價格: {gui_output_float * user_input_list["calculate_area"]}萬元"
        Output_label.configure(text=output_text, text_color="#5555FF")

    # 錯誤代碼
    elif(gui_output_float == -1):
        output_text = f"[ERROR -1]沒有可進行預測的資料"
        Output_label.configure(text=output_text, text_color="#CC0000")

    else:
        output_text = f"[ERROR ??]未知的錯誤代碼:{gui_output_float}"
        Output_label.configure(text=output_text, text_color="#CC0000")
```

錯誤範例: