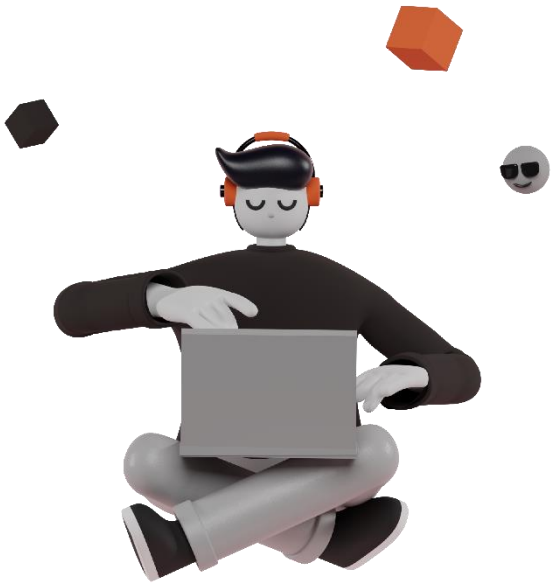
به نام خدا

گزارش کار پروژه نهایی

موضوع پروژه : مکان یابی شعبه جدید استارباکس

طراحی الگوریتم

تابستان 1402



پوریا حاجی قلی زاده

شماره دانشجویی : 40012507012

فهرست

مقدمه

بهینه‌سازی مکان احداث شعب و مکان‌یابی به عنوان یکی از موارد مهم در مدیریت کسب و کارها و برنامه‌ریزی‌های شهری مطرح است. در این حوزه، انتخاب مکان مناسب برای ایجاد شعبه یا واحد جدید، تأثیر مستقیم بر سودآوری و موفقیت کسب و کار دارد. از این رو، به‌دنبال روش‌هایی جهت بهینه‌سازی این فرآیند بوده‌ایم.

در این پروژه، به‌منظور انتخاب بهترین مکان برای احداث شعبه‌ی جدید در محیط شهری، از ترکیبی از الگوریتم K Nearest Neighbors (KNN) و معیارهای تخمین تقاضا و هزینه استفاده می‌کنیم. این پروژه به دنبال تعیین نقطه بهینه‌ای است که هم تخمین میزان تقاضا برای محصول یا خدمات مورد نظر را دارد و هم هزینه احداث شعبه در آن مکان به حداقل رسانده شود.

مراحل اصلی این پروژه شامل جمع‌آوری داده‌ها از مکان‌های مهم مانند مختصات شعب مترو، پمپ بنزین و استارباکس به همراه وزن‌های مربوط به آن‌ها می‌باشد. سپس با استفاده از تکنیک KNN ، نقاط احتمالی برای احداث شعبه جدید را در نظر می‌گیریم. با اعمال معیارهای مختلف مانند فاصله منهتن و وزن‌های مشخص شده برای هر مکان مهم، نقاط بهینه‌ای را انتخاب می‌کنیم که دارای ترکیبی از تخمین میزان تقاضا و هزینه احداث مناسبی باشند.

در ادامه، با محاسبه نسبت سود به هزینه برای هر نقطه بهینه، بهترین نقطه برای احداث شعبه جدید را انتخاب می‌کنیم. این روش نه‌تنها به ما در انتخاب بهترین مکان کمک می‌کند، بلکه نیز تأثیرات تخمین میزان تقاضا و هزینه را در این انتخاب مورد بررسی قرار می‌دهد.

در این پروژه، تلاش داریم تا با استفاده از روش‌های بهینه‌سازی و مدیریت مناسب مکان‌ها، کسب و کارها را به سمت رشد و پیشرفت هدایت کنیم و همچنین به توسعه‌ی پایدار شهرها و فضای شهری کمک نماییم.

ساختار کلی پروژه

ساختار کلی پروژه شامل چند مرحله است که به ترتیب اجرا می‌شوند. در زیر، ساختار کلی پروژه را شرح داده‌ام:

1. جمع‌آوری و آماده‌سازی داده‌ها:

در این مرحله، داده‌های مکان‌های مهم مانند مختصات شعب مترو، پمپ بنزین و استارباکس به همراه وزن‌های مربوط به آن‌ها جمع‌آوری و آماده‌سازی می‌شوند. این داده‌ها به عنوان ورودی‌ها برای مراحل بعدی استفاده می‌شوند.

2. محاسبه نقاط احتمالی:

با استفاده از تکنیکKNN (K Nearest Neighbors) ، نقاط احتمالی برای احداث شعبه جدید محاسبه می‌شوند. این مرحله شامل حلقه‌های تو در تو برای بررسی تمام نقاط احتمالی و محاسبه مرتبه زمانی فاصله و وزن‌ها برای هر مکان است.

3. انتخاب نقاط بهینه:

از میان نقاط احتمالی، نقاط بهینه بر اساس معیارهای تخمین میزان تقاضا و هزینه انتخاب می‌شوند. در این مرحله، معیارهای مختلفی مانند فاصله منهتن و وزن‌های مشخص شده برای هر مکان مهم در نظر گرفته می‌شوند.

4. محاسبه نسبت سود به هزینه و انتخاب بهترین نقطه:

نسبت سود به هزینه برای هر نقطه بهینه محاسبه شده و بهترین نقطه بر اساس این نسبت انتخاب می‌شود. این مرحله می‌تواند باعث انتخاب نقطه‌ای با ترکیب مناسب از تخمین میزان تقاضا و هزینه احداث شعبه شود.

5. نتایج و ارزیابی:

نقطه بهینه انتخاب شده و نتایج به دقت و جزئیات معرفی می‌شوند. ارزیابی تأثیر تخمین میزان تقاضا و هزینه در انتخاب نقطه بهینه نیز در این بخش صورت می‌پذیرد.

این ساختار کلی پروژه نمایان‌گر ترتیب اجرای مراحل و مهم‌ترین عملیات‌هایی است که در هر مرحله انجام می‌شود. هر یک از این مراحل به تصمیم‌گیری‌هایی در خصوص انتخاب مکان بهینه برای احداث شعبه جدید منجر می‌شود.

تفسیر کامل پروژه

```
import numpy as np
from sklearn.neighbors import NearestNeighbors
```

در این پروژه از دو کتابخانه "NumPy" و "sklearn.neighbors" استفاده شده است .

1. NumPy : این کتابخانه برای انجام محاسبات علمی و ریاضی به کار می‌رود. از طریق NumPy می‌توانید با آرایه‌ها و ماتریس‌ها کار کنید و عملیات مختلفی از جمله جمع، تفریق، ضرب، تقسیم و سایر محاسبات ریاضی را انجام دهید.
2. sklearn.neighbors :این کتابخانه بخشی از کتابخانه‌ی Scikit-Learn است و ابزارهای مختلفی را برای کار با مسائل مربوط به همسایگی و نزدیک‌ترین همسایه (KNN) ارائه می‌دهد. در اینجا، از کلاس "NearestNeighbors" برای پیاده‌سازی الگوریتم KNN جهت محاسبه نقاط احتمالی برای احداث شعبه جدید استفاده کردیم.

```
metro_locations = np.array([(1, 2), (1.5, 11), (3.5, 5), (4, 19), (8, 1), (12, 10)])
gas_station_locations = np.array([(2, 4), (5, 15), (6, 7), (10, 8)])
starbucks_locations = np.array([(2.5, 9), (3, 3), (7, 5), (8.5, 16)])
```

در کد ارائه شده، مختصات مکان‌های مختلف با استفاده از آرایه‌ها در NumPy تعریف شده‌اند. هر آرایه به ترتیب مختصات مکان‌ها را در خود ذخیره می‌کند. این مختصات به فرمت (x, y) هستند، که x نمایانگر مختصات افقی (عرض) و y نمایانگر مختصات عمودی (طول) مکان می‌باشد.

مختصات نقاط موجود در سه آرایه به شرح زیر هستند:

1. Metro Locations

این آرایه شامل مختصات مکان‌های مترو است. به عبارت دقیق‌تر، از هر مکان مترو شامل دو مختصات (x, y) به همراه وزن متناظر با آن مکان استفاده شده است.

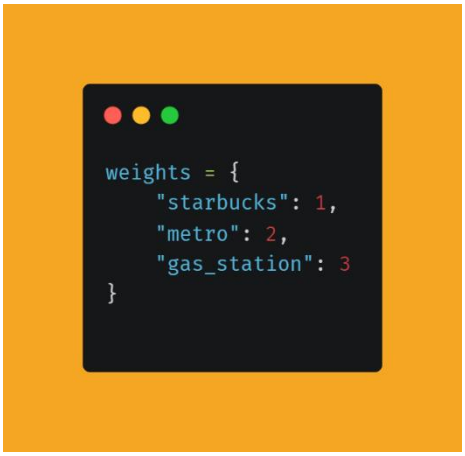
2. Gas Station Locations

این آرایه مختصات مکان‌های پمپ بنزین را نشان می‌دهد. به تشابه مکان‌های مترو، از هر مکان پمپ بنزین شامل دو مختصات (x, y) استفاده شده است.

3. Starbucks Locations

این آرایه مختصات مکان‌های شعب استارباکس را نمایش می‌دهد. همچنین، هر مکان شامل دو مختصات (x, y) به همراه وزن متناظر با آن شعبه است.

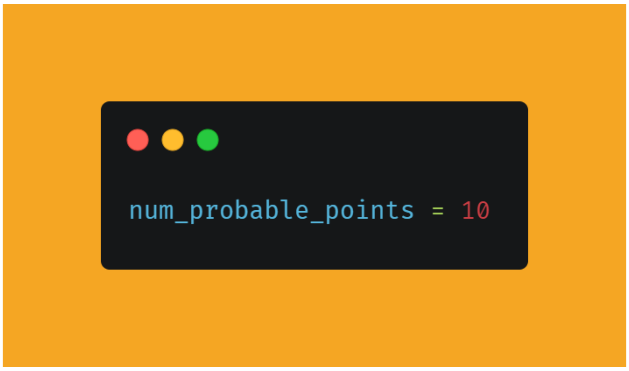
با استفاده از این آرایه‌ها، مختصات مکان‌ها و وزن‌های مرتبط با آن‌ها در پروژه برای محاسبه نقاط احتمالی جهت احداث شعبه جدید و محاسبات مرتبط با آن‌ها استفاده می‌شوند.



در این قسمت از کد، یک دیکشنری به نام "weights" تعریف شده است. این دیکشنری برای نگه‌داری وزن‌های مختلف مکان‌ها استفاده می‌شود. هر مکان (مترو، پمپ بنزین، استارباکس) وزن مخصوص به خود را دارد که به صورت کلید و مقدار در دیکشنری تعریف شده‌اند.

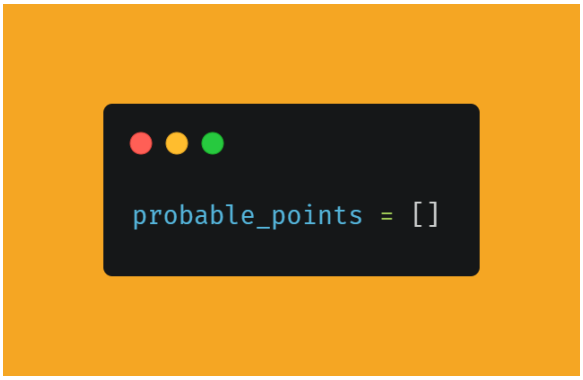
دیکشنری "weights" شامل سه کلید (key) به ترتیب "starbucks"، "metro" و "gas_station" است. مقدار مرتبط با هر کلید نمایانگر وزن مربوط به آن مکان است. به این ترتیب، برای محاسبه مجموع وزن‌ها برای هر نقطه احتمالی، می‌توان از وزن‌های مربوط به هر مکان استفاده کرد.

در داده های داده شده، وزن مکان‌های پمپ بنزین بیشتر از مترو و استارباکس است که نشان می‌دهد مکان‌های پمپ بنزین از اهمیت بیشتری در تصمیم‌گیری برای انتخاب نقاط بهینه برخوردار هستند.



در این بخش از کد، متغیری به نام "num_probable_points" تعریف شده است. این متغیر نشان‌دهنده تعداد نقاط احتمالی است که می‌خواهیم برای احداث شعبه جدید پیدا کنیم. با داشتن این تعداد، می‌توانیم از میان نقاط احتمالی که با استفاده از الگوریتم‌های مختلف محاسبه می‌شوند، تعدادی از بهترین نقاط را انتخاب کنیم که برای ایجاد شعبه جدید مناسب‌ترین‌ها باشند.

برای مثال، اگر "num_probable_points" برابر با 10 باشد، پس از محاسبه نقاط احتمالی، بهترین 10 نقطه از این نقاط انتخاب و در نظر گرفته می‌شوند که در نهایت یکی از آن‌ها برای احداث شعبه جدید انتخاب گردد. این مقدار می‌تواند تاثیرگذار بر کارایی و سرعت الگوریتم باشد و بسته به مسئله‌ی واقعی و نیازهای کسب و کار تعیین می‌شود.



در این قسمت از کد، یک لیست به نام "probable_points" تعریف شده است. این لیست به منظور ذخیره نقاط احتمالی برای احداث شعبه جدید استفاده می‌شود. در طول اجرای الگوریتم، نقاط احتمالی با محاسبه فاصله و وزن‌ها برای هر نقطه محاسبه شده و به این لیست اضافه می‌شوند.

به عبارت دقیق‌تر، به ازای هر مختصات (x, y) مورد نظر در بازه‌های مشخص، محاسبات لازم برای محاسبه مجموع فاصله‌ها و وزن‌ها برای آن نقطه انجام می‌شود و نتیجه به همراه مختصات آن نقطه به لیست "probable_points" اضافه می‌شود.

پس از انجام تمام محاسبات، این لیست حاوی اطلاعات مختلف مانند مختصات هر نقطه و مجموع فاصله‌ها و وزن‌ها برای آن نقطه خواهد بود. این اطلاعات در مراحل بعدی جهت انتخاب بهترین نقاط بهینه برای احداث شعبه جدید استفاده می‌شوند.

```
for x in range(0, 13):
    for y in range(0, 21):
        if (x, y) == (6, 7):
            continue  #
```

در این قسمت از کد، دو حلقه تو در تو (nested loop) برای محاسبه نقاط احتمالی جهت احداث شعبه جدید استفاده شده است. حلقه‌های تو در تو به ازای همه مقادیر ممکن از مختصات x و y در بازه‌های مشخص (0 تا 12 برای x و 0 تا 20 برای y) اجرا می‌شوند.

در هر ترکیب مقادیر x و y که در حلقه‌ها انتخاب می‌شوند، با استفاده از عبارت شرطی "if (x, y) == (6, 7):" بررسی می‌شود که آیا نقطه مورد نظر (7, 6) با مقادیر x و y مشخص شده در حلقه‌ها مطابقت دارد یا نه. اگر مطابقت داشته باشد، عبارت "continue" اجرا می‌شود.

دستور "continue" به معنی رد کردن اجرای بقیه قسمت‌ها از حلقه فعلی و ادامه اجرای حلقه بعدی است. بنابراین، اگر نقطه (7, 6) با مقادیر x و y مطابقت داشته باشد، اجرای محاسبات برای آن نقطه در این حلقه‌ها قطع خواهد شد و به محاسبه مختصات نقاط دیگر ادامه داده می‌شود. این کار باعث حذف نقطه (6, 7) از لیست نقاط احتمالی برای احداث شعبه جدید می‌شود.

علت استفاده این کد ، وجود مختصات (7, 6) در داده های مسئله و جز 10 نقاط احتمالی برگزیده بود با اینکار نقطه مشترک را در بازه دلخواه (10) از بین بردم.

```
        total_distance = 0
        for loc in metro_locations:
            distance = np.abs(x - loc[0]) + np.abs(y - loc[1])
            total_distance += distance * weights["metro"]
        for loc in gas_station_locations:
            distance = np.abs(x - loc[0]) + np.abs(y - loc[1])
            total_distance += distance * weights["gas_station"]
        for loc in starbucks_locations:
            distance = np.abs(x - loc[0]) + np.abs(y - loc[1])
            total_distance += distance * weights["starbucks"]

        probable_points.append(((x, y), total_distance))
```

این بخش از کد به منظور محاسبه مجموع فاصله‌ها و وزن‌ها برای هر نقطه احتمالی جهت احداث شعبه جدید استفاده می‌شود. برای هر مختصات (x, y) در محدوده‌های مشخص (0 تا 12 برای x و 0 تا 20 برای y)، این کد فواصل بین نقاط مکان‌های مترو، پمپ بنزین و استارباکس را محاسبه می‌کند و مجموع فاصله‌ها و وزن‌ها مرتبط با آن نقطه را محاسبه می‌کند.

برای هر مکان مترو، فاصله‌ای از نقطه (x, y) با مختصات مکان مترو محاسبه می‌شود. سپس این فاصله با وزن مترو ضرب شده و به مجموع فاصله‌ها و وزن‌ها اضافه می‌شود. همین محاسبات برای مکان‌های پمپ بنزین و استارباکس نیز انجام می‌شود.

در نهایت، مختصات (x, y) و مجموع فاصله‌ها و وزن‌ها برای آن نقطه به لیست "probable_points" اضافه می‌شوند. این لیست در مراحل بعدی جهت انتخاب بهترین نقاط احتمالی برای احداث شعبه جدید استفاده می‌شود.

در این بخش از کد، لیست "probable_points" که در مراحل قبلی ساخته شده است، به منظور مرتب‌سازی نقاط احتمالی براساس مجموع فاصله‌ها و وزن‌های مرتبط با آن نقاط مرتب می‌شود.

```
sorted_points = sorted(probable_points, key=lambda x: x[1])
```

- sorted_points: این متغیر جدیدی است که می‌خواهیم لیست نقاط احتمالی را بر اساس مجموع فاصله‌ها و وزن‌ها مرتب کنیم و در آن ذخیره کنیم.

- sorted(): این تابع پایتون برای مرتب‌سازی یک لیست و یا داده‌های قابل مرتب‌سازی دیگر استفاده می‌شود.

- probable_points: این پارامتر نشان‌دهنده لیست نقاط احتمالی است که در مراحل قبلی ساخته شده است و حاوی مختصات (x, y) و مجموع فاصله‌ها و وزن‌ها برای هر نقطه است.

- key=lambda x: x[1]: این قسمت از تابع sorted() به عنوان یک پارامتر به تابع داده می‌شود. این بخش مشخص می‌کند که بر اساس چه معیاری لیست نقاط احتمالی باید مرتب شود. در اینجا از یک

تابع مجهول (lambda function) استفاده شده است که بر اساس عنصر دوم هر تاپل (x[1])، یعنی مجموع فاصله‌ها و وزن‌ها، لیست نقاط احتمالی را مرتب می‌کند.

بنابراین، با اجرای این دستور، لیست نقاط احتمالی به ترتیب صعودی مرتب می‌شود، به این معنی که نقاطی که کمترین مجموع فاصله‌ها و وزن‌ها را دارند (بهترین نقاط) در ابتدای لیست قرار می‌گیرند.

```
best_probable_points = [point[0] for point in sorted_points[:num_probable_points]]
```

در این بخش از کد، با استفاده از لیستی که به ترتیب مجموع فاصله‌ها و وزن‌ها مرتب شده است، بهترین نقاط احتمالی برای احداث شعبه جدید انتخاب می‌شوند و در لیست جدیدی به نام "best_probable_points" ذخیره می‌شوند.

- best_probable_points: این لیست جدیدی است که قرار است بهترین نقاط احتمالی برای احداث شعبه جدید در آن ذخیره شوند.

- [point[0] for point in sorted_points[:num_probable_points]: این قسمت از کد از تکنیک نمایه‌گذاری لیست استفاده می‌کند تا مختصات (x, y) بهترین نقاط احتمالی (که در لیست sorted_points قرار دارند) را استخراج کند.

- [for point in sorted_points[:num_probable_points]: این حلقه به تعداد مقدار متغیر num_probable_points اجرا می‌شود (که از پیش تعیین شده است). این به این معنی است که تنها اولین تعداد num_probable_points نقطه از لیست sorted_points در نظر گرفته می‌شوند.

بنابراین، با اجرای این دستور، بهترین نقاط احتمالی برای احداث شعبه جدید از لیست مرتب‌شده انتخاب می‌شوند و در لیست جدید ذخیره می‌شوند.

این قسمت از کد به منظور نمایش بهترین نقاط احتمالی برای احداث شعبه جدید استفاده می‌شود. پس از انجام محاسبات و انتخاب بهترین نقاط احتمالی برای احداث شعبه جدید، این دستورات جهت نمایش این نقاط به کار می‌روند:

```
print("Best Probable Points:")
for point in best_probable_points:
    print(point)
```

- print("Best Probable Points:") این دستور یک خط متنی را چاپ می‌کند تا به عنوان عنوان یا برچسب نمایش بهترین نقاط احتمالی در خروجی ظاهر شود.

- for point in best_probable_points:: این حلقه برای هر نقطه در لیست بهترین نقاط احتمالی (best_probable_points) اجرا می‌شود.

- print(point) این دستور برای چاپ مختصات هر نقطه بهترین احتمالی به صورت جداگانه در خروجی اجرا می‌شود. به این ترتیب، مختصات هر نقطه به صورت جداگانه نمایش داده می‌شوند و خروجی شامل مختصات تمام بهترین نقاط احتمالی خواهد بود.

به این ترتیب، با اجرای این دستورات، نقاط بهترین احتمالی برای احداث شعبه جدید در خروجی چاپ می‌شوند.

در این قسمت از کد، مقداری به نام "k_neighbors" تعریف شده است. این متغیر نشان‌دهنده تعداد همسایه‌های نزدیک به هر نقطه احتمالی است که در مراحل بعدی برای اندازه‌گیری فاصله‌ها و محاسبه میزان تقاضا و هزینه از آن استفاده می‌شود.

به عبارت دقیق‌تر، اگر "k_neighbors" برابر با 2 باشد، این بدان معنی است که برای هر نقطه احتمالی، دو نقطه همسایه نزدیک‌تر به آن نقطه در نظر گرفته می‌شود تا فاصله و محاسبات مرتبط با آن‌ها محاسبه شوند.

تعیین این مقدار مهم است و می‌تواند تأثیر مستقیمی بر عملکرد و دقت الگوریتم داشته باشد. تعداد همسایه‌های بیشتر ممکن است دقت را افزایش دهد، اما همزمان زمان محاسباتی را نیز افزایش دهد. از طرف دیگر، تعداد همسایه‌های کمتر ممکن است دقت را کمتر کند، اما زمان محاسباتی را کاهش می‌دهد. بنابراین، انتخاب این مقدار به وابستگی به نیازها و محدودیت‌های واقعی مسئله بستگی دارد.

در این بخش از کد، یک لیست به نام "important_neighbors" تعریف شده است. این لیست به منظور ذخیره نقاط مکان‌های مهم و همسایه نزدیک به هر نقطه احتمالی جهت احداث شعبه جدید استفاده می‌شود.در مراحل بعدی از الگوریتم، به ازای هر نقطه احتمالی، محاسباتی انجام می‌شود تا دو نقطه مهم و همسایه نزدیک‌تر به آن نقطه پیدا شود. این نقاط مهم می‌توانند مختصات مکان‌های پمپ بنزین، مترو یا استارباکس باشند.

پس از انجام محاسبات، مختصات دو نقطه مهم و همسایه نزدیک‌تر به هر نقطه احتمالی و همچنین معلومات مرتبط با آن نقاط (مثل مجموع فاصله‌ها و وزن‌ها) در لیست "important_neighbors" ذخیره می‌شوند تا در مراحل بعدی از الگوریتم برای محاسبات استفاده شوند.

```
for point in best_probable_points:
    all_locations = np.concatenate((metro_locations, gas_station_locations, starbucks_locations))
    all_weights = np.concatenate((np.full(metro_locations.shape[0], weights["metro"]),
                                  np.full(gas_station_locations.shape[0], weights["gas_station"]),
                                  np.full(starbucks_locations.shape[0], weights["starbucks"])))

    nbrs = NearestNeighbors(n_neighbors=k_neighbors, algorithm='ball_tree').fit(all_locations)
    distances, indices = nbrs.kneighbors([point])

    weighted_distances = np.sum(distances * all_weights[indices[0]]) /
np.sum(all_weights[indices[0]])
    important_neighbors.append((point, weighted_distances))
```

این بخش از کد به منظور اندازه‌گیری فاصله‌ها و محاسبه مقادیر مرتبط با همسایه‌های نزدیک به هر نقطه احتمالی انجام می‌شود. در واقع، این بخش تاکید بر محاسبه مقدار بهینه‌سازی برای احداث شعبه جدید با توجه به فاصله و وزن‌ها در نقاط همسایه نزدیک به نقطه احتمالی دارد.

- for point in best_probable_points: این حلقه به ترتیب برای هر نقطه بهترین احتمالی اجرا می‌شود.

- all_locations = np.concatenate((metro_locations, gas_station_locations, starbucks_locations)): در این خط، تمام مختصات مکان‌های مترو، پمپ بنزین و استارباکس به هم متصل و در یک آرایه چندبعدی قرار می‌گیرند. این آرایه به نام "all_locations" نامگذاری می‌شود.

- all_weights = np.concatenate((np.full(metro_locations.shape[0], weights["metro"]), ...: در این خط، وزن‌های متناظر با مختصات مکان‌ها در "all_locations" به توجه به نوع مکان (مترو، پمپ بنزین یا استارباکس) به "all_weights" اضافه می‌شوند.

- nbrs = NearestNeighbors(n_neighbors=k_neighbors, algorithm='ball_tree').fit(all_locations): این دستور یک مدل مساحتی نزدیک‌ترین همسایه با استفاده از کتابخانه NearestNeighbors ایجاد می‌کند. تعداد همسایه‌ها به "k_neighbors" تعیین شده و الگوریتم محاسبه فضا (algorithm) به عنوان 'ball_tree' تعیین می‌شود.

- distances, indices = nbrs.kneighbors([point]): با استفاده از مدل ایجاد شده، فاصله‌ها و شاخص‌های نزدیک‌ترین همسایه‌ها برای نقطه فعلی محاسبه می‌شوند و در "distances" و "indices" ذخیره می‌شوند.

- weighted_distances = np.sum(distances * all_weights[indices[0]]) / np.sum(all_weights[indices[0]]): در این خط، مجموع فاصله‌ها ضرب‌شده با وزن‌های مرتبط با همسایه‌ها محاسبه می‌شود و سپس تقسیم بر مجموع وزن‌ها انجام می‌شود. این کار به دست آوردن مقدار بهینه برای فاصله نسبت به وزن‌ها کمک می‌کند.

```
sorted_important_neighbors = sorted(important_neighbors, key=lambda x: x[1])[:5]
```

در این بخش از کد، مراحلی برای انتخاب پنج نقطه بهترین احتمالی برای احداث شعبه جدید با توجه به مقادیر بهینه‌سازی محاسبه‌شده در مرحله قبل انجام می‌شود.

- sorted_important_neighbors: این یک لیست جدیدی است که قرار است پنج نقطه بهترین احتمالی برای احداث شعبه جدید را در آن ذخیره کند.

- sorted(important_neighbors, key=lambda x: x[1]): این دستور لیست "important_neighbors" را بر اساس مقادیر بهینه‌سازی (weighted_distances) مرتب می‌کند. عبارت `key=lambda x: x[1]` به معنای این است که مرتب‌سازی بر اساس مقدار دوم از هر تاپل در لیست انجام شود، یعنی مقدار weighted_distances.

- [:5]: این بخش باعث می‌شود تا تنها پنج نقطه اول از لیست مرتب‌شده "important_neighbors" در نظر گرفته شوند. چرا که ما قرار داریم پنج نقطه بهترین احتمالی را انتخاب کنیم.

به عبارت دقیق‌تر، با اجرای این دستور، پنج نقطه بهترین احتمالی برای احداث شعبه جدید بر اساس مقادیر بهینه‌سازی محاسبه‌شده از همسایه‌ها به لیست "sorted_important_neighbors" اضافه می‌شوند.

```
print("Top 5 Optimal Points based on Neighbor Weights and Manhattan Distance:")
for point in sorted_important_neighbors:
    print("Coordinates:", point[0], "- Weighted Distance:", point[1])
```

این بخش از کد برای نمایش بهترین پنج نقطه احتمالی برای احداث شعبه جدید با استفاده از مقادیر بهینه‌سازی محاسبه‌شده انجام می‌شود. این مقادیر شامل وزن‌های همسایه‌ها و محاسبه فاصله‌ها بر اساس فاصله منهتن می‌شوند.

- print("Top 5 Optimal Points based on Neighbor Weights and Manhattan Distance:"): این دستور یک خط متنی را چاپ می‌کند تا به عنوان عنوان یا برچسب نمایش بهترین نقاط احتمالی در خروجی ظاهر شود.

- for point in sorted_important_neighbors: این حلقه به ترتیب برای هر نقطه بهترین احتمالی از لیست "sorted_important_neighbors" اجرا می‌شود.

- print("Coordinates:", point[0], "- Weighted Distance:", point[1]): این دستورات به ترتیب مختصات و وزن محاسبه‌شده برای هر نقطه بهترین احتمالی را چاپ می‌کنند. با این کار، مختصات نقطه به همراه مقدار بهینه‌سازی محاسبه‌شده به خروجی چاپ می‌شوند. این اطلاعات نشان می‌دهد که کدام نقاط بهترین احتمال را برای احداث شعبه جدید دارند و چرا انتخاب آن‌ها بهینه است.

- important_neighbors.append((point, weighted_distances)): در این خط، مختصات نقطه احتمالی و مقدار بهینه محاسبه شده برای فاصله از همسایه‌ها به لیست "important_neighbors" اضافه می‌شوند.

وزن محاسبه‌شده در این مرحله از الگوریتم نشان‌دهنده اهمیت یا ارزش نقاط همسایه نزدیک به هر نقطه احتمالی است. در اینجا، با در نظر گرفتن مختصات مکان‌های مهم مانند مترو، پمپ بنزین و استارباکس، برای هر یک از این مکان‌ها وزنی تعریف شده است که نمایانگر اهمیت یا ارزش آن مکان نسبت به سایر مکان‌هاست.

در این مرحله از الگوریتم، برای محاسبه مقدار بهینه برای احداث شعبه جدید، از مفهوم وزن استفاده می‌شود. به عبارت دیگر، با توجه به فاصله نسبت به مکان‌های همسایه نزدیک، از مقادیر وزن متناسب

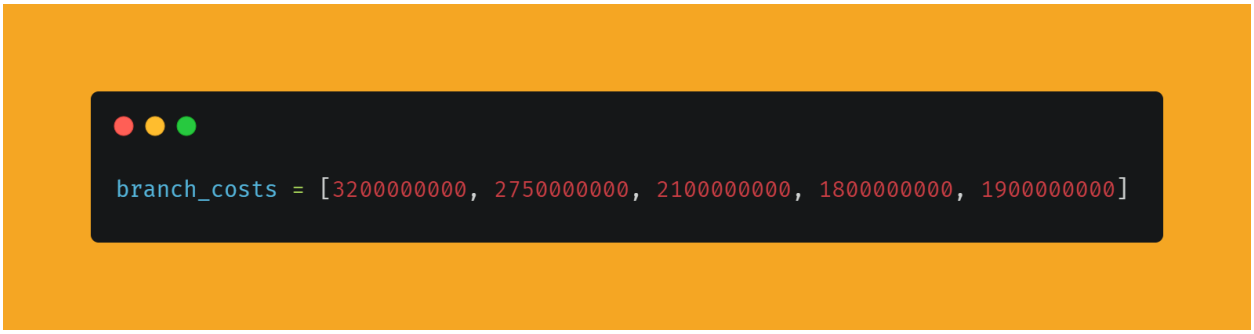
با اهمیت این مکان‌ها برای محاسبه مقدار بهینه برای احداث شعبه استفاده می‌شود. این کار به تصمیم‌گیری هوشمندانه‌تر برای انتخاب مکان بهترین احتمالی برای احداث شعبه جدید کمک می‌کند، زیرا نه تنها فاصله، بلکه همچنین اهمیت همسایه‌های نزدیک نیز در نظر گرفته می‌شود.

```
coffee_demands = [100, 120, 90, 150, 80]
```

این دستور در واقع یک لیست به نام "coffee_demands" ایجاد می‌کند که در آن مقادیر تخمین زده شده برای میزان تقاضای قهوه در هر یک از پنج نقطه احتمالی برای احداث شعبه جدید را نشان می‌دهد. این تخمین‌ها براساس اطلاعات موجود برای هر نقطه و تقاضای متوقع قهوه در آن نقطه انجام شده‌اند.

به طور مثال، برای نقطه احتمالی اول، تخمین میزان تقاضا قهوه ۱۰۰ قهوه در روز است. برای نقطه دوم، تخمین میزان تقاضا قهوه ۱۲۰ قهوه در روز و بقیه نقاط نیز به همین ترتیب دارای تخمین میزان تقاضا قهوه هستند.

این اطلاعات به تصمیم‌گیری برای انتخاب بهترین نقطه احتمالی برای احداث شعبه جدید اهمیت می‌دهد، زیرا می‌تواند تأثیر مستقیمی بر سودآوری و موفقیت شعبه داشته باشد.



```
branch_costs = [3200000000, 2750000000, 2100000000, 1800000000, 1900000000]
```

این دستور یک لیست به نام "branch_costs" ایجاد می‌کند که در آن مقادیر تخمین زده شده برای هزینه‌های احداث شعبه در هر یک از پنج نقطه احتمالی برای احداث شعبه جدید را نشان می‌دهد. این

تخمین‌ها بر اساس اطلاعات موجود برای هر نقطه و هزینه تقریبی احداث شعبه در آن نقطه انجام شده‌اند.

به طور مثال، برای نقطه احتمالی اول، تخمین هزینه احداث شعبه ۳،۲۰۰،۰۰۰،۰۰۰ تومان است. برای نقطه دوم، تخمین هزینه احداث شعبه ۲،۷۵۰،۰۰۰،۰۰۰ تومان و بقیه نقاط نیز به همین ترتیب دارای تخمین هزینه احداث شعبه هستند.

این اطلاعات به تصمیم‌گیری برای انتخاب بهترین نقطه احتمالی برای احداث شعبه جدید نیز اهمیت می‌دهد، زیرا هزینه‌ها به طور مستقیم تأثیرگذار بر سودآوری و موفقیت شعبه می‌باشند.

این دستور به منظور انتخاب بهترین نقطه احتمالی برای احداث شعبه جدید با توجه به ترکیب مقادیر تخمین زده شده برای تقاضای قهوه و هزینه‌های احداث شعبه در هر نقطه، اجرا می‌شود.

- np.array(coffee_demands): این بخش یک آرایه از تخمین میزان تقاضای قهوه در هر نقطه احتمالی را ایجاد می‌کند، که در لیست "coffee_demands" قرار دارد.

- np.array(branch_costs): این بخش نیز یک آرایه از تخمین هزینه‌های احداث شعبه در هر نقطه احتمالی را ایجاد می‌کند، که در لیست "branch_costs" ذخیره شده است.

- np.array(coffee_demands) / np.array(branch_costs): این بخش مقدار تقاضای قهوه به تقسیم بر هزینه‌های احداث شعبه را برای هر نقطه محاسبه می‌کند. این تقسیم نسبت تقاضا به هزینه را نشان می‌دهد.

- np.argmax(...): این بخش مکان نقطه با بیشترین مقدار نسبت تقاضا به هزینه را در آرایه محاسبه شده به دست می‌آورد. به عبارت دیگر، این بخش نقطه‌ای را که باعث داشته باشد تا نسبت تقاضا به هزینه بیشترین مقدار را داشته باشد (به عبارت دقیق‌تر، بیشترین عنصر در آرایه)، به عنوان بهترین نقطه احتمالی انتخاب می‌کند.

به طور خلاصه، این دستور بهترین نقطه احتمالی برای احداث شعبه جدید را بر اساس ترکیب میزان تقاضای قهوه و هزینه‌های احداث شعبه در هر نقطه انتخاب می‌کند.

```
best_point = sorted_important_neighbors[best_point_index][0]
print("Best Point for Branch Location:", best_point)
```

در این بخش از کد، نقطه بهترین احتمالی برای احداث شعبه جدید که با استفاده از ترکیب مقادیر بهینه‌سازی محاسبه‌شده انتخاب شده است، چاپ می‌شود.

- best_point_index: این متغیر نقطه‌ای را که بهترین احتمال را برای احداث شعبه دارد (با استفاده از ترکیب تخمین‌های میزان تقاضا و هزینه)، ذخیره می‌کند. این مقدار در مرحله قبل با استفاده از np.argmax(...) محاسبه شده است.

- sorted_important_neighbors[best_point_index][0]: این بخش نقطه مختصاتی از لیست بهترین نقاط احتمالی را که با استفاده از ترکیب میزان تقاضا و هزینه بهینه‌سازی شده است، استخراج می‌کند.

- print("Best Point for Branch Location:", best_point): این دستور نقطه بهترین احتمالی برای احداث شعبه جدید را با یک پیام چاپ می‌کند. مختصات این نقطه نشان داده می‌شود که به عنوان بهترین نقطه برای احداث شعبه جدید انتخاب شده است.

نتیجه گیری و خروجی کار

در این پروژه، ما با استفاده از مختصات موقعیت‌های مترو، پمپ بنزین و استارباکس در نقشه شهر، سعی داشتیم بهترین نقطه برای احداث یک شعبه جدید از یک زنجیره قهوه‌فروشی را تعیین کنیم. برای انجام این کار، مراحل زیر انجام شد:

1. ابتدا نقاط احتمالی را در نقشه محاسبه کرده‌ایم. این نقاط با توجه به فواصل تا موقعیت‌های مترو، پمپ بنزین و استارباکس محاسبه شده‌اند.

2. سپس از بین نقاط احتمالی، 10 نقطه بهینه را با ترکیب مقادیر بهینه‌سازی شده (مانند وزن‌ها) انتخاب کردیم.

3. در مرحله بعد، با استفاده از روش همسایگی نزدیک‌ترین همسایه (KNN)، مجموعه ای از نقاط همسایه نزدیک به هر یک از نقاط بهینه را محاسبه کردیم.

4. با محاسبه وزن‌ها بر اساس نقاط موجود و وزن‌ها، فاصله وزن‌دار میان نقاط بهینه و همسایه‌های نزدیک را محاسبه کردیم.

5. نهایتاً، از بین نقاط بهینه و محاسبه‌شده، نقطه‌ای را با ترکیب بهینه‌سازی سود و هزینه انتخاب کردیم که به عنوان بهترین نقطه برای احداث شعبه جدید در نظر گرفته شد.

در این پروژه از کتابخانه‌های `numpy` و `sklearn` استفاده کرده‌ایم تا عملیات محاسباتی و محاسبات KNN را انجام دهیم. این پروژه با ترکیب الگوریتم‌های مختلف و استفاده از مفاهیمی از جمله فاصله منهتن، وزن‌دهی و انتخاب بهینه، به ما کمک می‌کند بهترین موقعیت برای احداث شعبه جدید را در نظر گرفته و تصمیم‌گیری را بر اساس معیارهای مختلف بهبود دهیم.

تحلیل مرتبه زمانی

مرتبه زمانی کل پروژه می‌تواند به شرح زیر تحلیل شود:

1. حلقه دوتایی برای بررسی نقاط احتمالی:

- تعداد مجموعه‌های دوتایی `(x, y)` که به عنوان مختصات نقاط در نظر گرفته می‌شوند: 13 * 21 (این مقدار به عنوان متغیرهای حلقه داخلی و بیرونی به کار می‌رود)

- در هر حلقه داخلی، محاسبه فاصله نقطه مورد نظر تا موقعیت‌های مترو، پمپ بنزین و استارباکس (6 مرتبه محاسبه فاصله)

- پس از محاسبه مرتبه زمانی برای هر نقطه احتمالی، این نقاط به لیست `probable_points` اضافه می‌شوند. مرتبه زمانی این مرحله برابر است با: O(13 * 21 * 6) = O(1638)

2. مرتب‌سازی نقاط احتمالی:

- مرتب‌سازی لیست `probable_points` با توجه به مرتبه زمانی O(n * log(n)) که n برابر با تعداد نقاط احتمالی (270) است.

3. انتخاب بهترین نقاط احتمالی:

- انتخاب اولین 10 نقطه بهترین احتمال از لیست مرتب‌شده `probable_points` انجام می‌شود. مرتبه زمانی این مرحله به اندازه تعداد انتخاب‌هاست: O(10)

4. محاسبه همسایه‌ها و وزن‌دهی:

- برای هر یک از 10 نقطه بهترین احتمال، انجام محاسبات KNN و محاسبه وزن‌ها بر اساس نقاط موجود و وزن‌ها. مرتبه زمانی این مرحله برابر با O(10 * 2 * (مجموعه کل نقاط موجود)) است.

5. مرتب‌سازی همسایه‌های مهم:

- مرتب‌سازی لیست همسایه‌های مهم با توجه به فاصله وزن‌دار و مرتبه زمانی O(n * log(n)) که n برابر با تعداد نقاط احتمالی (10) است.

6. انتخاب بهترین نقطه بر اساس نسبت سود به هزینه:

- انتخاب بهترین نقطه بر اساس نسبت سود به هزینه از میان 5 نقطه بهینه‌ترین انجام می‌شود. مرتبه زمانی این مرحله برابر با O(5)

در کل، مرتبه زمانی کل پروژه به صورت تقریبی برابر است با:

O(1638 + 270 * 10 + log(270) * (2 * مجموعه کل نقاط موجود) + log(5 * 5))

که به طور خلاصه می‌توان آن را به O(n * log(n)) تقریب زد، که در اینجا n برابر با تعداد نقاط احتمالی (10) است.

پایان