# עבודת גמר - אופטימיזציה 88-782

רותם גזית, ת"ז 318446697

סמסטר חורף 2020

# תוכן העניינים

# 1 הקדמה

## 1.1 עולם הבעיה

נסיעה בפקקים הפכה להיות דבר שבשגרה למרבית האוכלוסייה בעולם המערבי. כיוון ששעות העבודה של רוב בתי העסק והחברות מסנוכרנות, נוצרים "זמני עומס" (rush hours) בהן הדרכים נמצאות בספיקה מקסימלית, מה שמוביל לפקקי תנועה.

פתרונות טכנולוגיים רבים עוסקים בתחום הזה. waze למשל מציעה למשתמשיה מסלולים אלטרנטיביים עמוסים פחות, אותם היא יכולה לחשב ע"ס משתמשים אחרים שנעים בצירים ומדווחים לה בחזרה מה המהירות הממוצעת בכל קטע. משתמש ב waze מקבל תשובה לשאלה "מה הדרך המומלצת ביותר בין X ל Y כרגע" על ידי כך שנוסעים אחרים זזים בצירים המרכזיים המקשרים את X ל Y כבר עכשיו.

מטרת הפרויקט היא לתת גישה אחרת לפתרון בעיית הנסיעה בפקקים. במקום לדעת **מהי** הדרך המומלצת **כרגע**, אני רוצה לדעת **מתי** לצאת כדי לנסוע כמה שפחות זמן בדרך שלי.

כלומר, אני מראש יודע מה הדרך המועדפת לנסיעה בין X ל Y, ורק רוצה לדעת מתי עדיף לי לנסוע בה. לצורך העניין - אני יודע מה הדרך הטובה ביותר מבחינתי לנסוע מביתי לאוניברסיטה (יכול להיות שהיא לא ניתנת לבחירתי, למשל אם משתמשים בתחבורה ציבורית), אבל המטרה שלי היא להיות כמה שפחות זמן על הכביש, ועל כן אני מבקש המלצה לזמן היציאה האידיאלי.

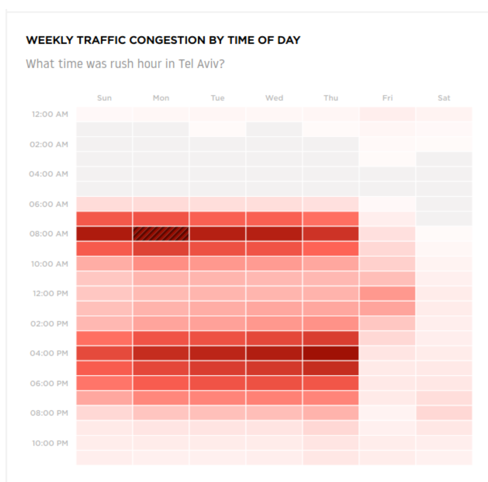
כמובן שמינימיזציה שכזו תהיה מאוד משעממת, כי סביר להניח שבאמצע הלילה כל הצירים יהיו ריקים וזמן הנסיעה בהם יהיה מינימלי. לכן היינו רוצים להוסיף זמן מינימלי ומקסימלי ליציאה בתור חסמים.

בסה"כ אני רוצה לדעת מה היא שעת היציאה שתיתן לי זמן נסיעה מינימלי בציר, תוך התנאי ששעת היציאה צריכה להיות בטווח זמנים מסוים.

הסיבה שאני רוצה לקבל מדד כזה נובעת מכך שבעבודה שלי אין לי שעת הגדרה קשיחה (היות ואני משתכר שכר גלובלי), אלא טווח זמנים שבו אני צריך להיות במשרד שלי. כיוון שאין לי סיבה מוגדרת להגיע בשעה $7^{\underline{30}}$ לעומת $7^{\underline{40}}$, אני פשוט מעדיף לנסוע כשאין פקקים.

בתל אביב מצב הפקקים חמור במיוחד[1]. תל אביב מדורגת במקום 9 מבין 240 הערים הפקוקות באירופה, ובמקום 21 מבין כלל ערי העולם שנבדקו.

בממוצע כל נסיעה מתארכת ב 46% מהזמן הנאיבי שלה (מרחק הנסיעה חלקי מהירות מקסימלית בכל אחד מהקטעים) בשל פקקים. התרשים הנ"ל מתאר את העומס הממוצע בכבישים בכל אחת משעות היום ובכל אחד מימות השבוע.




## 1.2 תכנית פעולה

את העבודה ניתן לחלק למספר שלבים:

1. מציאת מידע מתאים. צריך לשים לב שחשובים לנו הדברים הבאים:
   (א) דגימה של אותו נתיב הנסיעה לאורך זמן
   (ב) מספיק דגימות בזמנים שונים - שעות שונות, ימים שונים בשבוע, חודשים שונים, ימי חול וימי חופשה
   (ג) בעבור כל נסיעה - לדעת מתי התחילה וכמה זמן לקחה בפועל.
2. ביצוע רגרסייה על גבי המידע שתאפשר לנו לחזות את זמן הנסיעה ברמת דיוק מספיקה.
   כדי לבצע רגרסייה כזו נצמיד את המידע על הנסיעות לנתונים נוספים:
   (א) מה היה מזג האוויר באותו היום - רוחות, גשמים, טמפרטורות

[1] https://www.tomtom.com/en_gb/traffic-index/tel-aviv-traffic#statistics

(ב) הצמדת היום לימי חופשה לאומיים

3. ביצוע מינימיזציה על ה Regressor שנקבל.
המינימיזציה מניחה יום כלשהו עם כל נתוני המסגרת שלו, והתנאים שלה היא שזמן תחילת הנסיעה נמצא בטווח הזמנים המבוקש. המינימיזציה מחזירה את זמן תחילת הנסיעה המינימלי לפי הרגרסייה שלנו.

### 1.2.1 גיבוש מידע מתאים

התנאים המגבילים על המידע הנדרש המובאים לעיל הם תנאים מחמירים מאוד ורוב המידע בתחום זה איננו ציבורי.
בסופו של דבר, המידע שבו נעזרתי הוא מידע ציבורי על נסיעות מוניות ברחבי העיר ניו יורק[2]. המידע נאסף בשנת 2013 ומכיל 173,179,759 נסיעות מונית (29GB) בכלל רחבי העיר ועל גבי כל ימות השנה.
עבודה משמעותית של איתור בכלל הנתונים הללו צריכה להתבצע כדי לענות על תנאי 1א המתואר לעיל, כלומר לבודד נסיעות הקורות בין שתי נקודות ספציפיות על המפה.
את המידע הנ"ל מצליבים בקלות אל מול נתוני מזג אוויר[3] וימי חופשה לאומיים בארצות הברית[4]. על כלל העבודה הנ"ל יפורט בפרק ה data preperation.

### 1.2.2 אתגר המינימיזציה

לאחר קבלת מודל הרגרסייה המבוקש, בעיית המינימיזציה מעניינת כיוון שהיא משלבת שני קונספטים:

- מינימזציה של פונקצייה שהנגזרת שלה איננה ידועה
- מינימזציה של פונקציה עם אילוצים

במהלך הקורס למדנו כיצד ניתן למצוא מינימום לבעיות עם אילוצים בעזרת היכולת שלנו לגזור אותה. שיטות כמו KKT מאוד מוצלחות לפתרון בעיה זו.
כיוון שכאן הנגזרת לא ידועה היות והרגרסור שלנו איננו אנליטי, נמצא מינימום בעזרת שילוב שיטת Penalty Method יחד עם אחת השיטות שלמדנו בהרצאות הראשונות של הקורס - שיטת Nelder Mead (הידועה גם בשם downhill simplex method).
לאחר יישום השיטות הבסיסיות, ננסה לבצע היוריסטיקות שיאפשרו לנו למצוא מינימום גלובלי ולא לוקאלי.

## 2 הבניית המידע - data preperation

### 2.1 חילוץ המידע המתאים

המידע כאמור שעליו ננסה לפתור את הבעיה הוא פירוט נסיעות מונית ברחבי העיר ניו יורק על גבי כל שנת 2013.

| medallion | hack_license | ndor | rate_code | and_fw | pickup_datetime | dropoff_datetime | passenger_count | trip_time_in_secs | trip_distance | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CD483CB196E3505057E48C94FD89233B | 199EEEFA30FF9D8D0DDEAAE121107C8E | VTS | 1 | | 2013-01-15 10:42:00 | 2013-01-15 10:49:00 | 5 | 420 | 0.92 | -73.977074 | 40.750179 | -73.965347 | 40.754841 |
| 86845617E8D805CC175640E6701E2B9B | 8B608F94D85CE91ECA9D060C852C96B2 | VTS | 1 | | 2013-01-15 20:10:00 | 2013-01-15 20:17:00 | 3 | 420 | 1.47 | -73.971642 | 40.765968 | -73.978149 | 40.750393 |
| 92A5DA5E65D973B5F64C70113B091164 | BEB3C033E6D08F37F3415068B010D9EF | VTS | 1 | | 2013-01-14 19:58:00 | 2013-01-14 20:12:00 | 2 | 840 | 3.79 | -73.991051 | 40.734871 | -73.958488 | 40.763 |
| CFEF976DC97C18EFE560DED254BD6283 | 58816C3AEBF47213F74FBC8AFF915FCD | VTS | 1 | | 2013-01-16 00:06:00 | 2013-01-16 00:28:00 | 5 | 1320 | 6.05 | -73.981178 | 40.758041 | -73.992348 | 40.689629 |
| 8736738DE7D70316889BF9D38B7ED338 | D4E8282485A4228BB4872822C7EFEAA5 | VTS | 1 | | 2013-01-14 16:45:00 | 2013-01-14 16:49:00 | 1 | 240 | 0.94 | -73.970009 | 40.799686 | -73.96814 | 40.792007 |
| 744B4B8BF12CF57282B79A8DEB9B9B44 | B9DD7C80F538AD0BCE3D154021E733AD | VTS | 1 | | 2013-01-16 09:36:00 | 2013-01-16 09:58:00 | 1 | 1320 | 1.83 | -73.971291 | 40.748127 | -73.991966 | 40.748768 |
| B208E4003BAB08E4B77DDCCF722ABC | 4027AC0C230411EE4106E0071B7EDAB5 | VTS | 1 | | 2013-01-16 07:35:00 | 2013-01-16 07:52:00 | 1 | 1020 | 4.16 | -73.992706 | 40.758556 | -73.997398 | 40.714149 |
| 11922D84A5A91EF5F8317AF86C3E43E9 | E7BBC18C40F200AB9C881C5D72423D8D | VTS | 5 | | 2013-01-15 21:38:00 | 2013-01-15 21:39:00 | 1 | 60 | 0 | -74.035568 | 40.713505 | -74.035553 | 40.713509 |
| BE8510A528238ACE647199089ADF6B47 | A14986CCF33AF0266289D422ACB52132 | VTS | 1 | | 2013-01-13 16:59:00 | 2013-01-13 17:10:00 | 3 | 660 | 1.86 | -73.997864 | 40.745178 | -73.997215 | 40.725773 |
| 0CA06825E31D118DED987821A7EDA3FE | AFACF8312244283B1FC901B06D7979A0 | VTS | 1 | | 2013-01-16 08:13:00 | 2013-01-16 08:17:00 | 1 | 240 | 0.55 | -73.967865 | 40.755814 | -73.974487 | 40.75322 |

המידע בתצורתו המקורית מכיל את הנתונים הבאים:

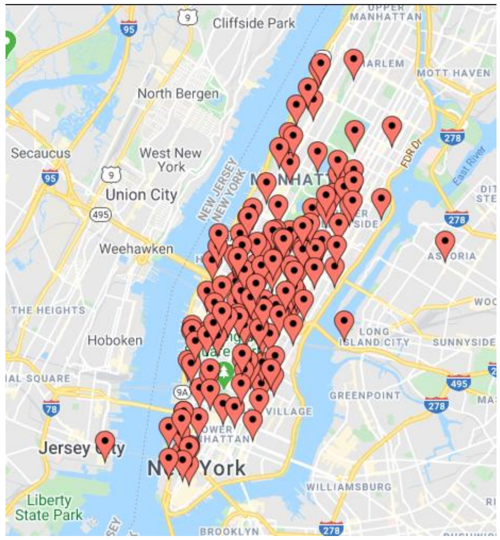
- נתוני מסגרת על הנסיעה: חברת המוניות, ו ID של הנסיעה המיוצג בעמודות:
  medallion hack_license vendor_id rate_code store_and_fwd_flag
- כמות הנוסעים במונית
  passenger_count
- נתוני זמן: זמן תחילת וסיום הנסיעה, והזמן בשניות של הנסיעה כולה
  trip_time_in_secs pickup_datetime dropoff_datetime
- המרחק שעברה המונית בזמן הנסיעה (במיילים)
  trip_distance
- קאורדינטות: קאורדינטת ההתחלה וקאורדינטת הסיום
  pickup_longitude pickup_latitude dropoff_longitude dropoff_latitude

[2] https://archive.org/details/nycTaxiTripData2013
[3] https://www.kaggle.com/selfishgene/historical-hourly-weather-data
[4] https://www.kaggle.com/gsnehaa21/federal-holidays-usa-19662020

המידע עצמו מכיל נתוני נסיעות מכל רחבי העיר, ועל כן לא מתאים לפתרון הבעיה שאנחנו מנסים לפתור (אופטימיזציה של שעת היציאה ע"ג מסלול בודד).

על כן תחילה נצטרך להתמקד באיזור מסוים בו יש לנו מספיק נסיעות לאורך השנה.

התרשים הבא מציג את מיקומן של 178 שורות רנדומליות מתוך המידע שלנו ומדגיש עד כמה הנסיעות מפוזרות ע"ג כלל העיר ניו יורק.




על מנת להתמקד באיזורים ספציפיים, נעזר בספרייה ש"מגרידה" (grid) קאורדינטות. למעשה כל קאורדינטה הופכת לקוד מיוחד שמייצג תא שטח. אנחנו נייצר תאי שטח בקוטר של בערך 250 מטר על מנת שבכל איזור כזה יהיו לנו מספיק נקודות.

הקוד של תהליך ה"הגרדה" נמצא בקובץ extract_points_of_interest.py. בשל כמות המידע הרבה שיש לנו (כ 30GB) אני נעזר בטכנולוגיית SPARK שלמעשה באופן מבוזר יכולה לעבד כמויות כאלה של מידע.

הספרייה שאיתה אני מבצע את ה"הגרדה" היא ספרייה של גוגל בשם openlocationcode. עטפתי אותה בפונקציה convert_coordinate_to_grid:

```
def convert_coordinate_to_grid(lat, long):
    try:
        return openlocationcode.encode(float(lat), float(long), 8)
    except:
        pass
```

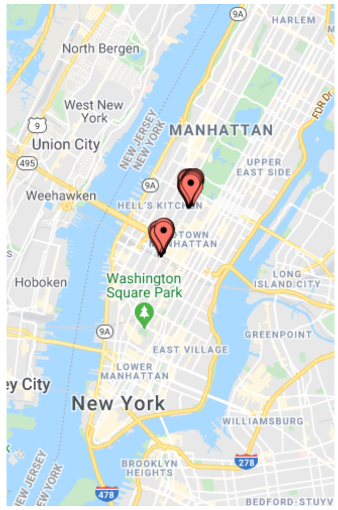
והחלתי אותה על ה Dataframe:

```
def add_grid_columns(df, columns_mapping=GRID_COLUMNS_MAPPING):
    convert_udf = F.udf(convert_coordinate_to_grid)
    for grid_column, lat_long_columns in columns_mapping.items():
        df = df.withColumn(grid_column, convert_udf(*lat_long_columns))
    return df
```

לאחר החלת ה GRID ע"ג כלל המידע, אני לוקח את 5 זוגות ה Grids (זה של תחילת הנסיעה וזה של סיום הנסיעה) שהופיעו הכי הרבה במידע, ואותם שומר בקבצים נפרדים (כדי שאוכל לאמן את המודל שלי עליהם בלבד). התוצר של התהליך נכתב לנתיב data/trip_data_grids.

יש לשים לב שאת כל ה dataset המקורי לא צירפתי לפרויקט מפאת גודלו ובמידה ורוצים להריץ את התהליך עליו יש להוריד אותו בנפרד. במקומו צירפתי דוגמית של החומרים לנתיב data/trip_data_sample.

ה GRIDS הכי פופולארים במידע היו 87G8Q225+, 87G8Q279+ , בהם יש כ 35 אלף נסיעות במהלך השנה. התרשים הבא מציג 120 נסיעות מתוכן על גבי המפה, וניתן לראות שאכן התהליך מיקד אותנו לאיזורי עניין בתוך ניו יורק ותואם את ה dataset אותו ניסינו להשיג במקור, כאשר יש שני ריכוזי נקודות, אחד עבור תחילת הנסיעה ואחד עבור סופה -



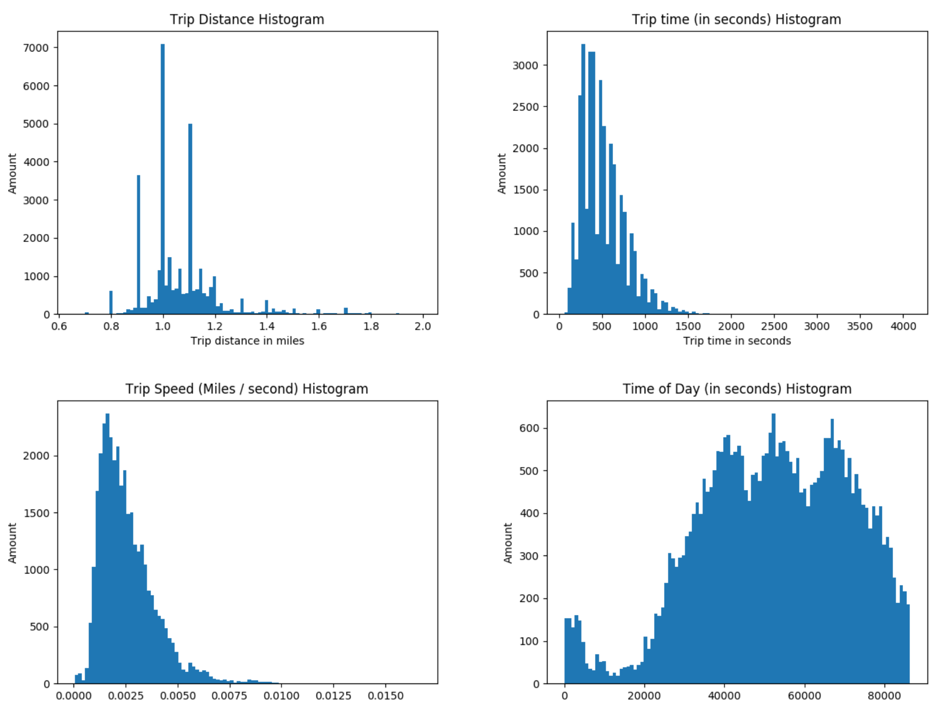

במהלך העבודה עד שלב זה ניסיתי דרכים נוספות לפתרון הבעיה:

- שימוש ב grid ממוקד יותר ברזולוצייה של 2 מטרים
האפשרות הזו נפסלה בגלל שהיא החזירה מעט מידי רשומות עבור תהליכי ההמשך. תהליך ה"הגרדה" לא מאפשר רזולוצייה שבין 250 המטרים ל2 המטרים ולכן נאלצתי להסתפק ברזולוצייה זו.

- זניחת השימוש ב grid והתבססות על מרחק אוקלידי בין נקודות
דרך זו קשה מאוד חישובית (למעשה היא $O(n^2)$ כאשר $n$ מספר הרשומות) ולכן נזנחה.

## 2.2 ויזואליזציה של המידע

בשלב זה ניתן להתחיל ולנתח את התפלגות המידע שקיבלנו.
הקוד ליצירת התרשימים הנ"ל נמצא בקובץ raw_data_stats.py




## 2.3 סיכום הפרק

1. הצלחנו לייצר dataset בן 35000 רשומות של נסיעות בין שני איזורים ספציפיים בעיר ניו יורק.
גיבשנו את המידע בעזרת תהליך סטנדרטי בתעשייה של הפיכת נ"צ לתא שטח.

2. המידע (לפחות מבחינה ויזואלית) מתפלג באופן נורמלי גם בזמני הנסיעה וגם באורך הנסיעה, ונראה שיש לנו נתונים מספיקים ע"ג כל שעות היום.
זה למעשה תואם את הציפיות הכלליות שלנו ממידע מסוג זה.

# 3 בניית המודל

## 3.1 הנדסת פיצ'רים מתאימים

כלל הקוד של הפרק הזה מופיע בנתיב python_code/model תחת הקובץ data_prep.py ותיקיית הקבצים features/

### 3.1.1 חילוץ מתוך המידע הקיים

השלב הראשון בעבודה שלנו הוא חילוץ הנתונים מתוך המידע שיצרנו בפרק הקודם.
מרבית הנתונים בטבלה המקורית אינם מעניינים אותנו. אני לא רוצה ללמוד על נתונים כמו כמות הנוסעים במונית או חברת המוניות, היות ואלו נתונים שלא רלוונטים לבעיה שאני מנסה לפתור (נסיעה ברכב פרטי). על כן מתוך המידע המקורי נשמור על 3 עמודות:

- זמן תחילת הנסיעה - pickup_datetime
- זמן הנסיעה הכולל בשניות - trip_time_in_secs
- מרחק הנסיעה הכולל במייל - trip_distance

מעמודת "זמן תחילת הנסיעה" נחלץ את הפיצ'ר המרכזי - זמן תחילת הנסיעה בשניות מאז תחילת היום

```
def format_dates(df):
    df = df.copy()
    df.pickup_datetime = pd.to_datetime(df.pickup_datetime, format='%Y-%m-%d %H:%M:%S')
    df['time_of_day'] = df.pickup_datetime.apply(lambda d: d.timestamp() % (24 * 60 * 60))
    df['DATE'] = df.pickup_datetime.dt.strftime('%Y-%m-%d')
    return df


def load_data(pickup_grid='87G8Q279+', dropoff_grid='87G8Q225+', select_cols=SELECT_COLS):
    data_path = os.path.join(os.path.abspath(TRIP_DATA_GRIDS_PATH), f'trip_data_{pickup_grid}_{dropoff_grid}.csv')
    df_grid = pd.read_csv(data_path)
    df_grid = df_grid[select_cols]
    df_grid = format_dates(df_grid)
    return df_grid
```

בקובץ python_code/model/features/date_parts.py מתבצע פירוק של עמודת הזמן לגורמיה - אנחנו לוקחים מתוך הזמן שלושה מאפיינים - היום בשבוע, היום בחודש והחודש.
נשים לב שלמרות שבמרבית התהליכים המקבילים היינו מחלצים גם את השנה, כאן כל האירועים קרו באותה השנה ולכן לא נרצה לעוות את הלמידה שלנו בשל כך.

```
def add_date_parts(df):
    df = df.copy()
    attrs = ['Dayofweek', 'Day', 'Month', ]
    for attr in attrs:
        df[f'pickup_datetime{attr.lower()}'] = getattr(df.pickup_datetime.dt, attr.lower())
    return df
```

### 3.1.2 מזג אוויר

את הנתונים ניתן להצמיד לנתוני מזג אוויר התואמים את זמני הנסיעה. המידע של מזג האוויר הגיע מפרויקט Kaggle ומכיל נתונים שעתיים של מזג האוויר בניו יורק (ובערים נוספות) ע"ג תקופת זמן של מספר שנים.
המידע מגיע ע"ג מספר קבצים נפרדים, לפי המדד - טמפרטורה, לחות, כיוון הרוח, עוצמת הרוח, לחץ ברומטרי, ותיאור מילולי של מזג האוויר.
הטבלה הבאה מציגה דוגמת חומר מטבלת הטמפרטורות וטבלת התיאור המילולי. אפשר לשים לב שהטמפרטורות נתונות בקלווין, והזמנים נתונים בזמן UTC (ולכן נצטרך להזיז אותם לזמן ניו יורק)

| datetime | New York | Tel Aviv District |
|---|---|---|
| 1Oct12 – 30Nov17 | 251 – 310 | 271 – 321 |
| 2012-10-01 13:00:00 | 288.22 | 305.47 |
| 2012-10-01 14:00:00 | 288.24767617 | 304.31 |
| 2012-10-01 15:00:00 | 288.326939663 | 304.281841331 |
| 2012-10-01 16:00:00 | 288.406203155 | 304.238014609 |
| 2012-10-01 17:00:00 | 288.485466648 | 304.194187887 |
| 2012-10-01 18:00:00 | 288.564730141 | 304.150361165 |
| 2012-10-01 19:00:00 | 288.643993634 | 304.106534443 |
| 2012-10-01 20:00:00 | 288.723257127 | 304.06270772 |
| 2012-10-01 21:00:00 | 288.80252062 | 304.018880998 |
| 2012-10-01 22:00:00 | 288.881784113 | 303.975054276 |
| 2012-10-01 23:00:00 | 288.961047606 | 303.931227554 |
| 2012-10-02 00:00:00 | 289.040311099 | 303.887400832 |

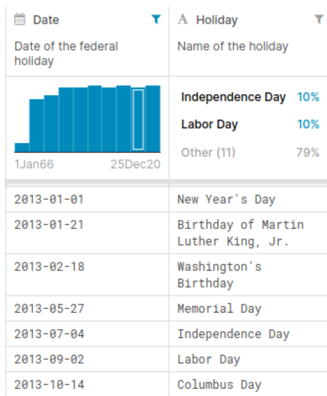
| datetime | New York | Tel Aviv District |
|---|---|---|
| 1Oct12 – 30Nov17 | sky is clear 26%<br>broken clouds 14%<br>Other (35) 61% | sky is clear 48%<br>few clouds 18%<br>Other (25) 34% |
| 2012-10-01 14:00:00 | few clouds | sky is clear |
| 2012-10-01 15:00:00 | few clouds | sky is clear |
| 2012-10-01 16:00:00 | few clouds | sky is clear |
| 2012-10-01 17:00:00 | few clouds | sky is clear |
| 2012-10-01 18:00:00 | few clouds | sky is clear |
| 2012-10-01 19:00:00 | few clouds | sky is clear |
| 2012-10-01 20:00:00 | few clouds | sky is clear |
| 2012-10-01 21:00:00 | few clouds | sky is clear |
| 2012-10-01 22:00:00 | few clouds | sky is clear |
| 2012-10-01 23:00:00 | few clouds | sky is clear |
| 2012-10-02 00:00:00 | few clouds | sky is clear |

הקובץ python_code/model/features/weather.py לוקח את נתוני מזג האוויר ומצמיד אותם בזה אחר זה לתוך הטבלה המקורית שלנו.
החיבור הוא טריוויאלי, המורכבות היחידה היא שינוי איזור הזמן.
על גבי החומרים השעתיים, אנחנו מחשבים גם 3 עמודות - הנתון המינימלי, המקסימלי והממוצע לאותו היום. הפונקציה הנ"ל מעשירה את המידע השעתי בנתונים הללו:

```python
def calculate_daily_mean(weather_dataset, file_name, date_column='datetime'):
    if weather_dataset.dtypes[file_name] == np.float:
        mean_column, min_column, max_column = f'{file_name}_daily_mean', f'{file_name}_daily_min', f'{file_name}_daily_max'
        means = weather_dataset[[date_column, file_name]]
        means[mean_column] = means[file_name]
        means[min_column] = means[file_name]
        means[max_column] = means[file_name]
        means = means.groupby([date_column], as_index=False).agg(
            {mean_column: 'mean', min_column: 'min', max_column: 'max'})
        means = means[[date_column, mean_column, min_column, max_column]]
        weather_dataset = means
    return weather_dataset
```

### 3.1.3 חגים לאומיים בארצות הברית

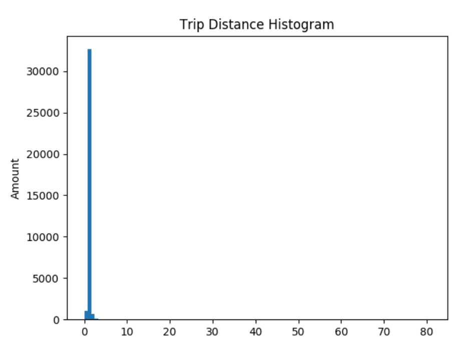
אחד הפרמטרים המשמעותיים (אינטואיטיבית) על מצב התנועה בכבישים הוא האם מדובר ביום עבודה או ביום חופש לאומי, מתוך הנחה שבימי החופש התנועה מתנהגת אחרת
(מצד אחד - פחות תנועה למקומות העבודה, ומצד שני עלייה בתיירות).
המידע עליו אני מתבסס מגיע גם הוא מפרויקט Kaggle ומכיל את כל החגים הלאומיים בארצות הברית בין השנים 1966 ל 2020.
כאן מעניין אותנו רק קיום בינארי - האם יום מסויים הוא יום חופש לאומי או לא.



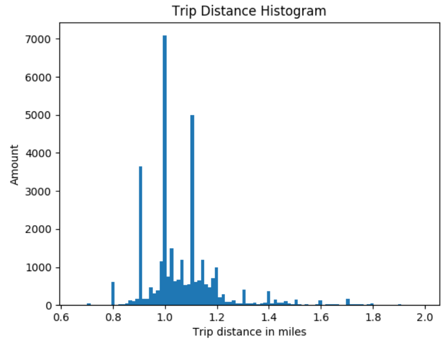

גם כאן ההצמדה פשוטה מאוד ומתבצעת בקובץ python_code/model/features/holidays.py

### 3.1.4 סינון אנומליות במידע

מתוך המידע שמתקבל נרצה לסנן מספר תופעות לא מוסברות שככל הנראה נובעות מהסנסורים שהפיקו במקור את הנתונים על מוניות.
ספציפית, מניתוח בסיסי ניתן לראות שלעתים יש רשומות עם אורך נסיעה ארוך או קצר מידי.
התרשים הבא הוא היסטוגרמת אורכי הנסיעה, כפי שהוצגה בפרק הקודם, רק ללא סינון על מרחק נסיעה מקסימלי:




ניתן לראות שבאמת מרבית הנסיעות מרוכזות בטווח קטן מאוד, אך ככל הנראה ישנן נסיעות (פיקטיביות?) שאורכן גדול מאוד.
כדי להתמודד עם אנומליות כאלו, נסתכל על האחוזונים השונים של מרחק הנסיעה. אחוזון 99% של מרחק הנסיעה הוא 2, ואחוזון $0.01\%$ הוא $0.65$.
על כן כדרך התמודדות נתבונן רק על רשומות בהן מרחק הנסיעה $d$ מקיים $0.65 \leq d \leq 2$. בזכות הסינון הנ"ל נקבל היסטוגרמה הגיונית יותר:




את כל תהליך הוספת הפיצ'רים מבצעת הפונקציה data_prep שנמצאת תחת הקובץ python_code/model/data_prep.py: היא טוענת את המידע, מנקה אותו ומצמידה את כל הפיצ'רים המתוארים לעיל.

## 3.2 אימון המודל

### 3.2.1 אימון

המודל הנבחר לצורך ביצוע המשימה הוא Random Forest Regressor וזאת מפאת פשוטו בניתוח בעיות מהסוג הנ"ל. כיוון שליבת הפרויקט איננה במימוש המודל (אלא בביצוע מינימיזציה עליו בסופו של התהליך), לא אכביר בכל הניסיונות השונים בהרצת המודל עד להגעה לתוצאות המוצגות כאן. הרצת המודל מתבצעת בקובץ python_code/model/model_learning.py

אנחנו מאמנים את המודל על 80% מהמידע, עם 40 עצים שונים, שכל אחד מאומן על 50% מהפיצ'רים שלנו. הגבלתי את מספר הדגימות בכל עלה בעץ ל 3.

רשימת ה Features המלאה שלנו בשלב זה:

| feature | description | feature | description |
|---|---|---|---|
| trip_time_in_secs | המשתנה התלוי – זמן הנסיעה | temperature_daily_min | טמפרטורה – מינימום ליום הנסיעה |
| trip_distance | אורך הנסיעה | temperature_daily_max | טמפרטורה – מקסימום ליום הנסיעה |
| time_of_day | זמן מתחילת היום בשניות | wind_direction_daily_mean | כיוון חח – ממוצע ליום הנסיעה |
| holiday | האם מדובר ביום חופש | wind_direction_daily_min | כיוון חח – מינימום ליום הנסיעה |
| humidity_daily_mean | לחות – ממוצע ליום הנסיעה | wind_direction_daily_max | כיוון חח – מקסימום ליום הנסיעה |
| humidity_daily_min | לחות – מינימום ליום הנסיעה | wind_speed_daily_mean | מהירות חח – ממוצע ליום הנסיעה |
| humidity_daily_max | לחות – מקסימום ליום הנסיעה | wind_speed_daily_min | מהירות חח – מינימום ליום הנסיעה |
| pressure_daily_mean | לחץ ברומטרי – ממוצע ליום הנסיעה | wind_speed_daily_max | מהירות חח – מקסימום ליום הנסיעה |
| pressure_daily_min | לחץ ברומטרי – מינימום ליום הנסיעה | pickup_datetimedayofweek | יום בשבוע |
| pressure_daily_max | לחץ ברומטרי – מקסימום ליום הנסיעה | pickup_datetimeday | יום בחודש |
| temperature_daily_mean | טמפרטורה – ממוצע ליום הנסיעה | pickup_datetimemonth | חודש |

### 3.2.2 מדידת איכות המודל

על תוצאת המודל הרצנו את הבדיקות הבאות (אל מול המידע עליו התאמנו ואל מול 20% מהחומרים שלא נכנסו לאימון):

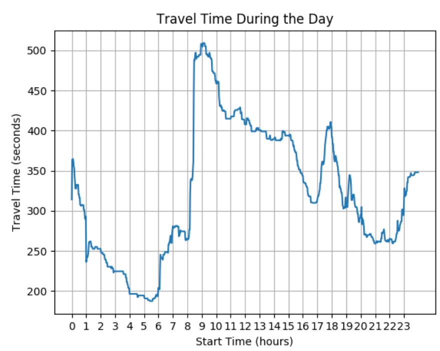
- הפרש זמנים בין הזמן האמיתי לזמן המנוחש ע"י המודל: $|T_{real} - T_{predict}|$
  - תוחלת ההפרשים
  - סטיית התקן של ההפרשים
  - חציון ההפרשים
  - אחוזון 95 של ההפרשים
  - הפרש מינימלי והפרש מקסימלי
- ציון $R^2$ למתאם המודל על התוצאות האמיתיות. ציון זה מופיע לנו על ה train, ה test וה OOB ( out of bag score, באופן כללי חישוב פרדיקציה על החומרים שלא נכנסו בתהליך הרנדומלי שבוחר חומרים לאימון)

| name | train | test |
|---|---|---|
| l1 - mean | 66.913623 | 97.797677 |
| l1 - std | 70.332301 | 98.777692 |
| l1 - quantile | 48.590813 | 74.054876 |
| l1 - 95% percentile | 187.413907 | 269.010393 |
| l1 - min | 0.002254 | 0.006385 |
| l1 - max | 2984.010486 | 2764.043598 |
| r2 | 0.857826 | 0.707454 |
| r2 - oob | 0.697148 | NaN |

התוצאות שקיבלנו אינן מושלמות (למשל ה $R^2$ על חומרי ה test נמוך ממה שהיינו רוצים), אבל ניתן לראות שאחוזון 95% של ההפרשים עומד על 269 שניות (פחות מ $4.5$ דקות), שזה לחלוטין לא רע בהתחשב בכמות הפיצ'רים הקטנה שבה השתמשנו, וזה מספיק כדי שנוכל להתקדם הלאה.

מן הסתם, עבודה מאומצת שמטרתה המוצהרת היא לייצר מודל טוב יותר הייתה מניבה תוצאות מוצלחות יותר, אך כיוון שמדדי המודל אינם ליבת העבודה, אני מרשה לעצמי לעצור במצב הזה.

התרשים הבא מתאר את זמן הנסיעה המנוחש (בהפרשים של 100 שניות) על פני כל שעות היום בתאריך $22/4/2013$:
ניתן לראות מהתרשים כמה דברים מעניינים:




1. המודל מצליח לזהות את התבנית של זמני עומס ושפל, באופן דומה לאופן שבו הגיוני לחשוב עליהם (בלילה - אין עומס, וישנם שני גלי עומס, בבוקר ולקראת ערב)

2. הפונקציה הנוצרת איננה רציפה, ויש בה הרבה מאוד "רעשים".
מינימיזציה על פונקציה רועשת כזו תביא אותנו למינימום לוקאלי מאוד ולא נכון.

### 3.2.3 רציפות הפונקציה הנוצרת

על מנת שנוכל לבצע מינימיזציה מוצלחת על הפונקצייה הנ"ל, נרצה לנקות אותה מרעשים ולהפוך אותה לרציפה.
לשם כך נעזר בטענה הבאה:

טענה 1. תהי $f:\mathbb{R}\to\mathbb{R}$ פונקציה אינטגרבילית (רימן) בתחום $[a,b]$. יהי $\epsilon>0$, הפונקציה $g_\epsilon(x)=\int_{x-\epsilon}^{x+\epsilon} f(x)dx$ רציפה לכל $x\in(a,b)$.

הוכחה. יהי $\delta>0$. נשים לב שמתקיים:

$$\begin{aligned}|g_\epsilon(x)-g_\epsilon(x+\delta)| &= \left|\int_{x-\epsilon}^{x+\epsilon} f(x)dx-\int_{x-\epsilon+\delta}^{x+\epsilon+\delta} f(x)dx\right| \\ &= \left|\int_{x+\epsilon}^{x+\epsilon+\delta} f(x)dx-\int_{x-\epsilon}^{x-\epsilon+\delta} f(x)dx\right|\end{aligned}$$

היות והפונקציה $f(x)$ אינטגרבילית בתחום, היא גם חסומה בו. לכן ניתן להניח כי $|f(x)|\le M$, ולכן:

$$\begin{aligned}\left|\int_{x+\epsilon}^{x+\epsilon+\delta} f(x)dx-\int_{x-\epsilon}^{x-\epsilon+\delta} f(x)dx\right| &\le \int_{x+\epsilon}^{x+\epsilon+\delta} Mdx+\int_{x-\epsilon}^{x-\epsilon+\delta} Mdx \\ &= 2\cdot\delta\cdot M\end{aligned}$$

מכלל הסנדוויץ', בגלל ש $0 \xrightarrow{\delta\to 0} 2\cdot\delta\cdot M$ מתקיים $|g_\epsilon(x) - g_\epsilon(x+\delta)| \xrightarrow{\delta\to 0} 0$ כלומר $g_\epsilon(x) \xrightarrow{\delta\to 0} g_\epsilon(x+\delta)$.
לכן $g_\epsilon(x) \xrightarrow{t\to x} g_\epsilon(t)$ לכל $x \in (a,b)$, משמע $g_\epsilon$ רציפה. $\square$

הערה. כמובן שגם הפונקציה $\frac{1}{2\epsilon}\int_{x-\epsilon}^{x+\epsilon} f(x)dx$ היא רציפה.
תהי החלוקה $\{x_{n_i}\}_{i=0}^{n}$ כאשר $x-\epsilon = x_{n_0} < x_{n_1} < \ldots < x_{n_n} = x+\epsilon$ חלוקה ל $n$ מקטעים. אזי:

$$\frac{1}{2\epsilon}\int_{x-\epsilon}^{x+\epsilon} f(x)dx = \lim_{n\to\infty}\frac{1}{2\epsilon}\cdot\sum_{i=1}^{n} f(x_{n_i})\cdot\frac{x+\epsilon-x+\epsilon}{n}$$

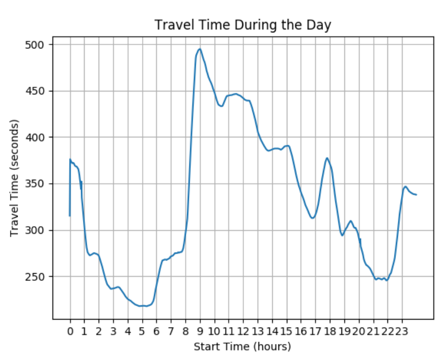
$$= \lim_{n\to\infty}\frac{1}{n}\cdot\sum_{i=1}^{n} f(x_{n_i})$$

על כן אם נבחר $\epsilon > 0$, יתקיים $\frac{1}{2\epsilon}\int_{x-\epsilon}^{x+\epsilon} f(x)dx \xleftarrow{n\to\infty} \frac{1}{n}\cdot\sum_{i=1}^{n} f(x_{n_i})$, ולכן נוכל לקחת $n$ מספיק גדול כדי לקבל קירוב לאינטגרל הנ"ל.

לפי הטענה, מתוך הנחה שפונקציית ה predict היא אינטגרבילית בתחום הערכים של יום מסוים, נוכל "להחליק" אותה ע"י קירוב האינטגרל הנ"ל לכל נקודה.
בקובץ python_code/model/pretty_predict.py אנחנו מיישמים פונקציית פרדיקציה אלטרנטיבית שמחשבת את **ממוצע** הפרדיקציות בסביבת הנקודה:

```
def predict(df, model, row, time_of_day=None, buffer=1000, buffer_samples=21):
    columns = df.columns
    time_of_day_index = list(columns).index('time_of_day')
    rows = []
    time_of_day = time_of_day or row[time_of_day_index]
    for i in np.linspace(max(time_of_day - buffer, 0), min(time_of_day + buffer, 60 * 60 * 24), buffer_samples):
        row = row.copy()
        row[time_of_day_index] = i
        rows.append(row)
    rows = pd.DataFrame(rows, columns=columns)
    preds = model.predict(rows)
    return preds.mean()
```

הפונקציה מקבלת את המידע והמודל שיצרנו, יחד עם שורה ספציפית עליה אנחנו רוצים לבצע predict, היא מייצרת שורות בטווח ובכמות המוגדרות כפרמטר (ברירת המחדל כאן היא בטווח של $\epsilon = 1000_{sec}$ לייצר $n = 21$ דגימות), ומחשבת את התוחלת של ה predict עליהן.
כאשר נצייר גרף של אותו היום המתואר לעיל בשימוש בפונקציה החדשה, נקבל:




### 3.2.4 סיכום הפרק

1. הצלחנו לייצר מודל שנותן ניבוי מוצלח של הבעיה שאותה אנחנו מנסים לפתור.
המודל רחוק מלהיות אידיאלי אך מדגים את הבעיה היטב.

2. הצלחנו לייצר מתוך המודל פונקציה שתאפשר לנו לבצע מינימיזציה, זאת בעזרת "החלקה" של הפונקציה כפי שיוצאת מהמודל עצמו.

# 4 מינימיזציה

## 4.1 הגדרת הבעיה

לאחר שקיבלנו את ה Regressor המבוקש, הגדרת הבעיה כרגע היא כזו:

יהי $R:\mathbb{R}^p \to \mathbb{R}$ ה Regressor שלנו לאחר ההחלקה שלו, כאשר $p$ מספר הפיצ'רים שלנו. בהנתן זמן ביום $t \in [0, 86400]$, ווקטור $\bar{x} \in \mathbb{R}^{p-1}$ המכיל את יתר הפיצ'רים שלנו. נשים לב שכולם קבועים ביחס לאותו היום. $R(t, \bar{x})$ הוא זמן הנסיעה הצפוי.
בעיית המינימיזציה שלנו היא כזו:

$$\begin{aligned} &\min R(t, \bar{x}) \\ &\quad \text{S.T.} \quad t \in [t_{min}, t_{max}] \\ &\quad \phantom{\text{S.T.}} \quad \bar{x} = \bar{x}_{today} \end{aligned}$$

העובדה שהפיצ'ר היחידי שמשתנה במהלך היום הוא $t$ חשוב מאוד כי אנחנו למעשה נקבע את $\bar{x}$ לחלוטין לאותו היום ונבצע מינימיזציה עם אילוצי אי שיוויון על $t$ בלבד. למעשה זה שקול ליצירת פונקציה $R_{\bar{x}}:\mathbb{R} \to \mathbb{R}$ שמקבלת זמן $t$ ביום ומחזירה את זמן הנסיעה הצפוי. במקרה הזה בעיית האופטימיזציה שלנו ממוקדת עוד יותר:

$$\begin{aligned} &\min R_{\bar{x}}(t) \\ &\quad \text{S.T.} \quad t \in [t_{min}, t_{max}] \end{aligned}$$

את $R_{\bar{x}}$ ניצור בעזרת הפונקציה הבאה (שמקבלת את המודל שלנו ואת $\bar{x}$ ומחזירה לנו פונקציה חדשה)

```python
c = Cache(1e9)
def get_regressor(model, x, predict):
    regressor = lambda t: predict(model, x, t)
    regressor = c.memoize(regressor)  # In order to speed up the evaluation - Cache is used on the regressor
    return regressor
```

## 4.2 מימוש

כיוון והפונקציה $R_{\bar{x}}$ שלנו היא רציפה אך הנגזרת שלה איננה ידועה, המימוש שלנו יתבסס על שילוב בין שתי שיטות: שיטת Penalty Method שתאפשר לנו להזניח את העיסוק באילוץ, ושיטת Nelder Mead שתאפשר לנו למצוא מינימום לוקאלי ללא שימוש בנגזרות.
מימוש השיטות נמצא בנתיב python_code/minimization

### 4.2.1 מימוש Penalty Method

השיטה אומרת את הדבר הבא. תהי פונקציה $f(x)$ שלה אנחנו רוצים למצוא מינימום תחת האילוצים $c_i(x) \leq 0 \quad \forall i \in I$, אזי נוכל למצוא מינימום לפונקציה ללא אילוצים:

$$P_k(x) = f(x) + \sigma_k \sum_{i \in I} \max(0, c_i(x))^2$$

וכאשר $\sigma_k \overset{k\to\infty}{\longrightarrow} \infty$ , יתקיים $\min f(x) \overset{k\to\infty}{\longrightarrow} \min P_k(x)$ כאשר הכוונה היא שהמינימום של הפונקציה $P$ ללא אילוצים ישאף למינימום תחת האילוצים של $f$.
במקרה שלנו, לפונקציה $R_{\bar{x}}(t)$ ישנם שני אילוצים:

$$c(t) = \begin{pmatrix} t - t_{max} \\ t_{min} - t \end{pmatrix}$$

ולכן:

$$P_k(t) = R_{\bar{x}}(t) + \sigma_k \cdot \max(0, t - t_{max})^2 + \sigma_k \cdot \max(0, t_{min} - t)^2$$

הפונקציה הזו עוטפת את $R_{\bar{x}}$ ומחזירה את $P_k(t)$ :

```python
def penalty_predict(regressor, time_of_day, time_of_day_min, time_of_day_max, factor=1):
    res = regressor(time_of_day)
    res += factor * max([time_of_day - time_of_day_max, 0]) ** 2
    res += factor * max([time_of_day_min - time_of_day, 0]) ** 2
    return res


def get_penalty_wrapper(regressor, time_of_day_min, time_of_day_max):
    return lambda t, factor: penalty_predict(regressor, t, time_of_day_min, time_of_day_max, factor)
```

כעת, בכל איטרציה נגדיל את $\sigma_k$ פי 10, ונמצא מינימום לפונקציה הזו. נעצור כאשר ההפרש בין המינימום שמצאנו לקודם קטן מטווח שגיאה שנגדיר, או כשנגיע לרף האיטרציות שנגדיר. הפונקציה הבאה מיישמת את התהליך:

```python
def minimize_penalty_in_boundry(regressor, t0, tmin, tmax, factor_iterations, minimizer, error=1e-5):
    print(f'minimize using {t0}, {tmin}, {tmax}, {factor_iterations}')
    res = None
    last_res = None
    counter = 0
    penalty = get_penalty_wrapper(regressor, tmin, tmax)
    while (res is None or last_res is None or last_res[0] - res[0] > error) and counter < factor_iterations:
        last_res = res
        res_ = minimizer(f=lambda t: penalty(t, 10 ** counter), x0=[res[1] if res else t0])
        res_value = penalty(res_, 10 ** counter)
        res = [res_value, res_[0]]
        counter += 1
    return res
```

### 4.2.2 מימוש Nelder Mead

אלגוריתם Nelder Mead או בשמו האחר Downhill Simplex הוא אלגוריתם שנלמד בקצרה בכיתה. הוא אלגוריתם למציאת מינימום לפונקציה רב מימדית ללא שימוש בנגזרות.

הסיבה שהאלגוריתם הזה נבחר לעבודה הוא בגלל הפופולאריות שלו וההתכנסות המהירה שלו. היותו מיושם בהרבה מאוד ספריות בפייתון אפשרה לי לבדוק את נכונות המימוש בהשוואה מולם.

האלגוריתם עובד כך:

תהי $f : \mathbb{R}^n \to \mathbb{R}$. בכל רגע נתון הוא מחזיק $x_1, ..., x_{n+1}$ נקודות. יהיו $\alpha > 0,\ \gamma > 1,\ 0 < \rho < 0.5,\ \sigma < 1$

1. מיין את הנקודות לפי $f$ : $f(x_1) \leq ... \leq f(x_{n+1})$
2. בדוק האם תנאי העצירה מתקיים
3. חשב את $x_0 = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i$
4. שלב **Reflection**: חשב $x_r = x_0 + \alpha(x_0 + x_{n+1})$. במידה ומתקיים $f(x_1) \leq f(x_r) \leq f(x_n)$, החלף $x_{n+1} = x_r$ וחזור ל 1.
5. שלב **Expansion**: אם $f(x_r) < f(x_1)$ אז חשב $x_e = x_0 + \gamma(x_r - x_0)$. אם $f(x_e) < f(x_r)$ אז החלף $x_{n+1} = x_e$ וחזור ל 1, אחרת $x_{n+1} = x_r$ וחזור ל 1
6. שלב **Contraction**: חשב $x_c = x_0 + \rho(x_{n+1} - x_0)$ . אם $f(x_c) < f(x_{n+1})$ אז החלף $x_{n+1} = x_c$ וחזור ל 1
7. שלב **Shrink**: החלף את הנקודות: $\forall i > 1 \ \ x_i = x_1 + \sigma(x_i - x_1)$ , וחזור לשלב 1

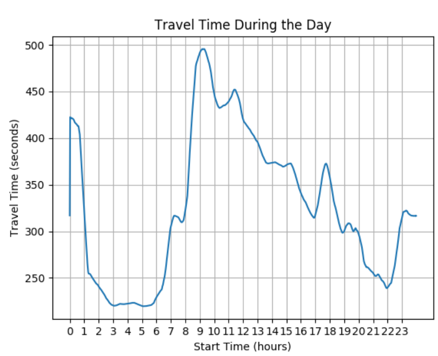
כדי לייצר את $x_1, ..., x_{n+1}$ ההתחלתיים, ניקח את נקודת ההתחלה $x$ בשילוב עם ווקטורי היחידה:

$$\forall 1 \leq i \leq n : x_i = x + e_i, \quad x_{n+1} = x$$

המימוש של התהליך הנ"ל מופיע בקובץ python_code/minimization/nelder_mead.py.

## 4.3 הדגמת שימוש

התרשים הבא מתאר את זמן הנסיעה המנוחש ביום רנדומלי. במהלך השנה. הקובץ app.py מדפיס תחילה את התרשים הנ"ל, ולאחר מכן מוצא מינימום בקטע שבין השעות 10 ל 12.

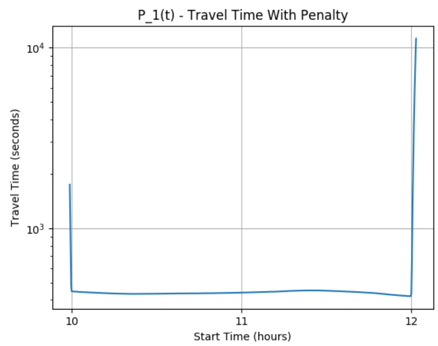



את מינימום זמן הנסיעה בטווח $t \in [10 \cdot 60 \cdot 60, 12 \cdot 60 \cdot 60]$, נקודת ההתחלה ממנה התחלנו לחפש הוא 0 שניות מתחילת היום, ואנחנו מבצעים 10 איטרציות על $P_k(t)$ (כלומר, מגדילים בכל פעם את $\sigma_k$ פי 10)

המינימום שנמצא הוא בשעה $10\frac{19}{} = 10.31111$, שם זמן הנסיעה עומד על 433.6610 שניות. מהתבוננות בתרשים ניתן לראות שזה באמת מינימום לוקאלי בטווח הזמנים, אך קיים מינימום גלובאלי בדיוק בקצה האיזור (בשעה 12).

ננסה לראות כיצד פרמטרים שונים להרצה יתנו תוצאות אחרות. כאשר שינינו את נקודת ההתחלה להיות בשעה 13 (אינטואיטיבית - אחרי הטווח שאנחנו מנסים לאתר בו, ולא לפניו) - קיבלנו מינימום בשעה $12^{\underline{00}} \sim 11.9984$, שם התקבל זמן נסיעה של $420.3702$ שניות.

נשים לב שלמרות שהמינימום האמיתי נמצא ב 12 בדיוק, Penalty Method מתקשה להגיע לתשובה הנכונה בגלל הקרבה של הערך לפונקציית ה"ענישה" שלנו. אם נצייר את $P_1(t)$, היא נראית כך:




ניתן להתגבר על כך ע"י זה שנבצע בדיקה יזומה על הקצוות של הטווח ונשווה למינימום שמצאנו.
נבצע את הניסוי הבא:

- נבחר 10 תאריכים רנדומליים
- בעבור כל אחד - נחפש את המינימום בין השעות 10 ל 12 בשיטת Nelder Mead פעמיים - פעם אחת עם 0 בתור נקודת ההתחלה, ופעם שנייה עם השעה 13 בתור נקודת ההתחלה.
- נריץ מינימיזציה נאיבית על הטווח - נרוץ על טווח השעות בין 10 ל12 בקפיצות של 100 שניות בכל פעם, וניקח בסופו של דבר את הטווח שהחזיר ערך מינימלי בפונקציה. השיטה הזו מאוד לא יעילה אך תחזיר את המינימום הגלובלי בטווח הנבחר עם שגיאה של עד 50 שניות לכל היותר.

מצורפות תוצאות הניסוי. בכחול - המינימום ש Nelder Mean מצא שהוא גם המינימום הגלובלי לפי השיטה ה"נאיבית". כל זוג עמודות מייצג שיטה אחרת למציאת המינימום, כאשר argmin הוא הזמן (בשעות) ליציאה, וmini הוא זמן הנסיעה המשוער (בשניות).

| date | T0 = 0 argmin | T0 = 0 min | T0 = 13 argmin | T0 = 13 min | NAIVE argmin | NAIVE min |
|---|---|---|---|---|---|---|
| 18 / 12 | 10.0000 | 616.1254 | 10.0000 | 616.1254 | 10.0000 | 616.1254 |
| 27 / 4 | 10.0000 | 361.1666 | 10.9208 | 331.9949 | 10.9167 | 332.2417 |
| 14 / 10 | 10.0000 | 590.3122 | 10.0000 | 590.3122 | 10.0000 | 590.3122 |
| 29 / 1 | 11.0308 | 371.1742 | 11.0142 | 371.1605 | 11.0000 | 371.3076 |
| 20 / 2 | 10.6467 | 451.7602 | 10.6361 | 452.2766 | 10.6389 | 452.0178 |
| 12 / 6 | 12.0000 | 660.1254 | 11.9983 | 659.9052 | 11.9722 | 660.6095 |
| 2 / 7 | 11.4642 | 515.6491 | 11.5074 | 515.8966 | 11.5556 | 515.7216 |
| 11 / 3 | 11.1997 | 383.5529 | 12.0000 | 387.9942 | 11.1944 | 383.5529 |
| 17 / 6 | 12.0000 | 549.5601 | 11.9986 | 549.5601 | 11.9722 | 552.9747 |

# 5 סיכום

בסופו של דבר, הפתרון המוצג כאן בעבודה הוא פתרון לבעיה שלא למדנו במהלך השיעורים בקורס - מינימיזציה של פונקציות חסרות נגזרת עם אילוצים. הפתרון אליו הגעתי בסופו של דבר יצא פשוט ומוצלח.

כשחיפשתי בעיה עבור הפרויקט, רציתי מאוד לשלב תהליך למידה מקדים ועליו לבצע את המינימיזציה. כפי שניתן לראות, זו איננה בעיה פשוטה. בשלב מתקדם של הפרויקט הבנתי שהפונקציה שהמודל שלי מייצר לא מאפשרת לי למצוא מינימום אמיתי בגלל שהיא מאוד "רועשת", והייתי צריך לחשוב על טריק כדי להחליק אותה. התהליך המוצג כאן בעבודה הוא התוצאה של החשיבה הזו. כמובן שהוא איננו בא בחינם, היות וכל קריאה לפונקציה גורמת לעיבוד של פי 20 יותר חומרים (היות ואני לוקח עוד 20 דגימות בסביבת הנקודה כדי לחשב את הממוצע ביניהן).

שיטת Penalty היא שיטה שמאוד אהבתי גם במהלך הקורס, היות והיא נראית כמו פתרון "עוקף" מוצלח מאוד לבעיית האילוצים. היות והצלחתי לעשות רדוקציה לבעיה בממימד אחד, יחסית פשוט להתגבר על נושא רגישות הקצוות בתחום, כי יש בסה"כ שניים לבדיקה. כמובן שבפונקציה רב מימדית עם אילוצים מורכבים יותר זה לא עניין כזה פשוט.

אפשר לראות מתוצאות הניסוי הקטן שבוצע בפרק האחרון גם את המורכבות של הבעיה המקורית אותה רציתי לפתור. ניתן לשים לב שהזמן המינימלי ליציאה שונה כמעט בכל אחד מהימים, למרות שתמיד הסתכלתי על אותו טווח זמנים.

במהלך העבודה על הפרויקט ניסיתי מגוון שיטות רחב עד שהגעתי לשיטת הפתרון כפי שנראתה כאן:

1. החלפת שיטת המינימיזציה ל Golden Scaling תוך וויתור על Penalty Method - הפתרון עבד, אך זמן הריצה היה גבוה כיוון ש Golden Scaling מתכנסת באופן ליניארי בלבד.
2. ניסיונות נוספים בהחלקת הפונקציה
3. מגוון פיצ'רים נוסף אותו ניסיתי לשיפור המודל
4. שיטות נוספות להבאת מידע רלוונטי לבעיה, והרצת ניסויים על Datasetים אחרים עד שהחלטתי להשתמש במידע המוצג כאן.