МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»
Кафедра КМАД

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №6

Виконав студент:

Омельніцький Андрій Миколайович
Група: КН-120А

Перевірила:

Ольга Василівна Костюк

Харків, 2023 г.

Зміст

Глава 1

Мета роботи

Вивчення математичної моделі на основі шаблонів, оптимізація моделі за різними критеріями.

Порядок виконання:

1. Використовуючи дані певного магазину, що надані викладачем за варіантом, навести повну характеристику даних, побудувати графіки.
2. Побудувати за даними (за виключенням останнього тижня) шаблони й модель на основі шаблонів.
3. Отримати прогнозні значення за моделлю на останній тиждень даних, порівняти з фактичними даними, зробити висновки.
4. Запропонувати критерій якості моделі. Вказати, яким чином цей критерій можна оптимізувати з метою підвищення якості прогнозування.
5. Виконати оптимізацію побудованої моделі, використовуючи в якості оптимізаційної змінної обмежену передісторію (змінну кількість тижнів) для побудови шаблонів моделі.
6. Виконати оптимізацію побудованої моделі, використовуючи в якості оптимізаційної змінної обмежену передісторію (змінну кількість тижнів) для адаптивного обчислення коефіцієнта.
7. Виконати оптимізацію побудованої моделі, використовуючи в якості шаблону найбільш схожий з попередніх днів.
8. Порівняти якість прогнозування отриманих моделей в пп. 5-7,
9. За результатами оптимізації надати рекомендації щодо побудови оптимальної прогнозної моделі.
10. Усі результати, отримані в ході виконання роботи, занести до звіту. Зробити висновки.

Глава 2

Основна частина

2.1. Пункт 1

Розглянемо данні магазину №530. За період від 2019-10-11 до 2020-01-28. Надалі й будемо розглядати данні цього магазину.
На рис. 2.4 представлений графік автокореляційної функції (АКФ) транзакцій магазину за період, що досліджується. З аналізу корелограми АКФ видно, що часовий ряд транзакцій містить явну циклічну компоненту з періодом k = 24.
Зобразимо графіки що харектиризують данні.

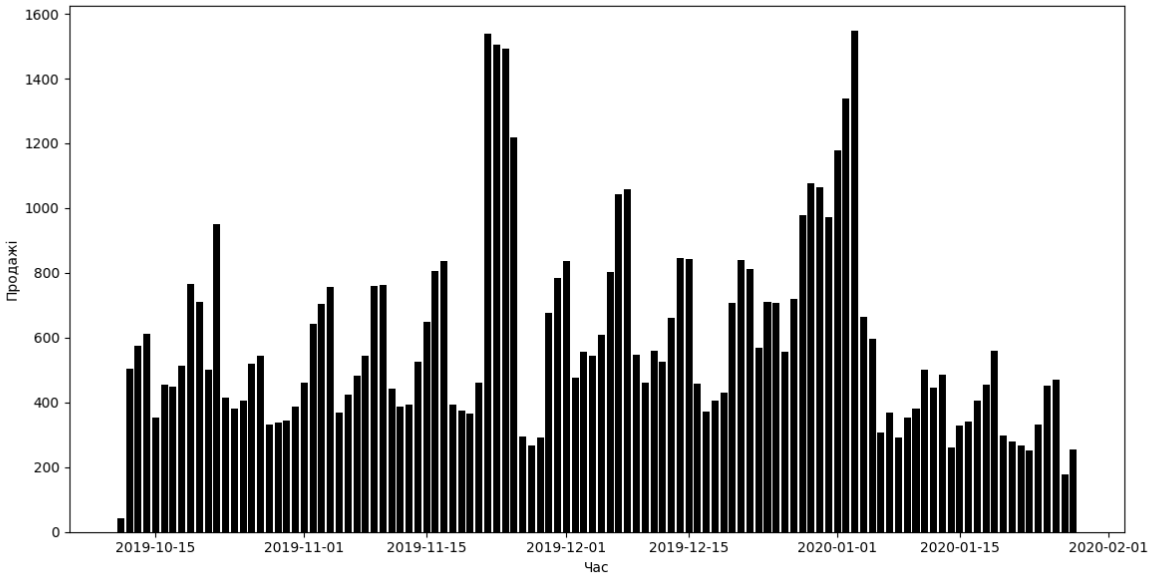

Рис. 2.1. Погодинна динаміка транзакцій магазину

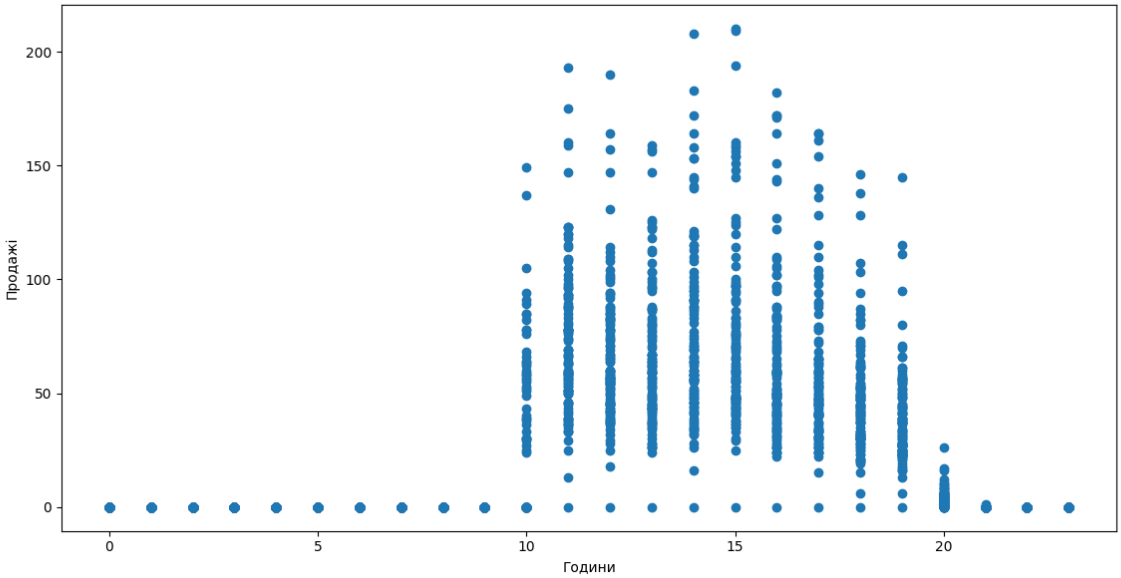

Рис. 2.2. Денний потік покупців за досліджуваний період

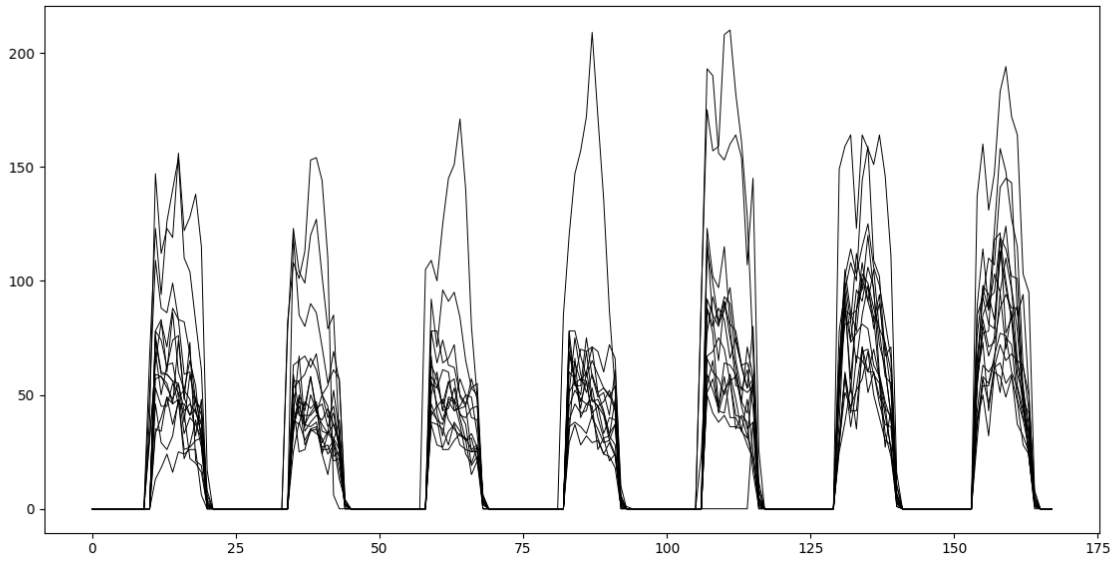

Рис. 2.3. Тижневий потік покупців за досліджуваний період.

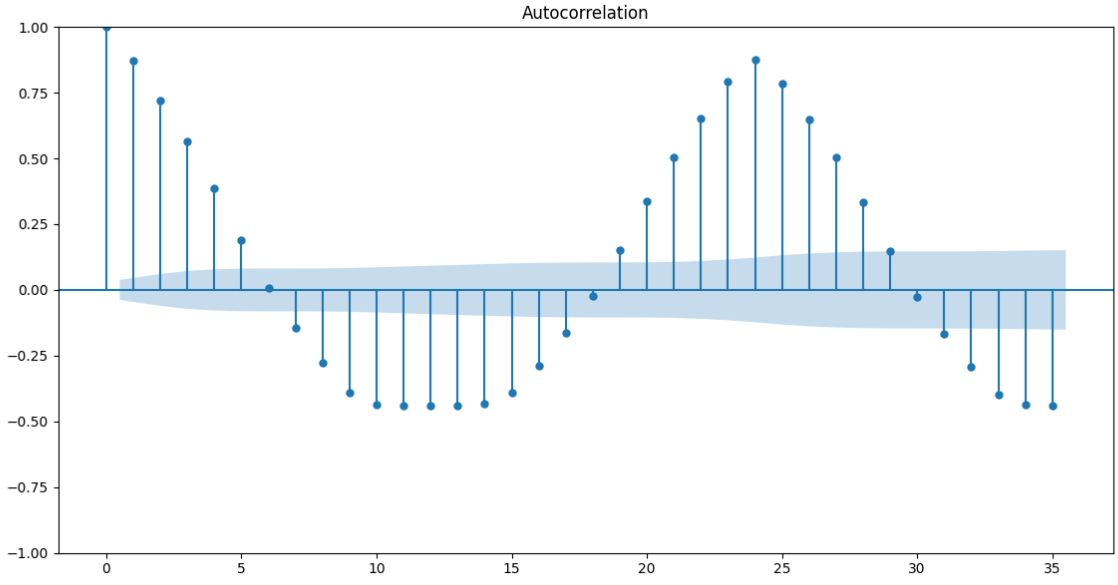

Рис. 2.4. Графік АКФ транзакцій за досліджуваний період.

2.2. Пункти 2-3

Побудуємо шаблони та зобразимо їх графічно. Побудуємо два вида шаблонів за медіаною та за середнім. Але надалі будемо використовувати побудовані за медіаною.

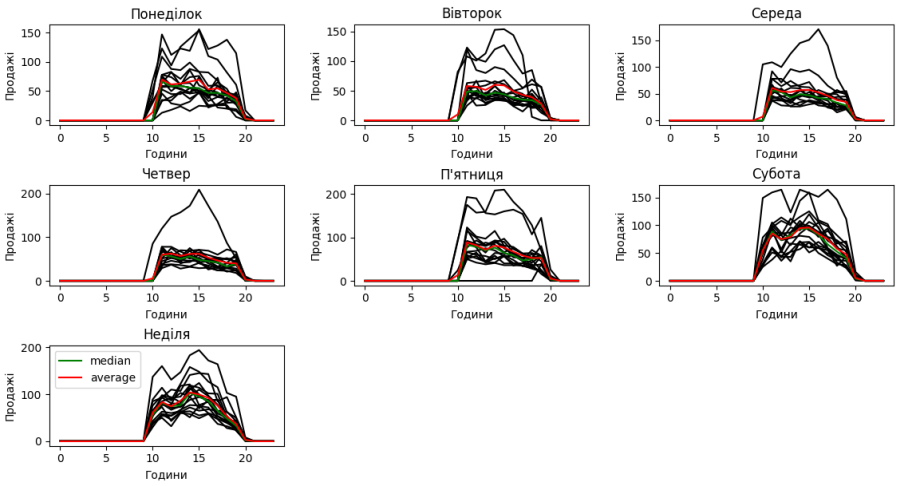

Рис. 2.5. Тижневий потік покупців за досліджуваний період та шабони побудовані за ними.

Отже для перевірки моделі побудуємо шаблон для нашого магазину на основі усіх данних окрім останнього тижня. А прогноз зробимо для вівторка останнього тижня.

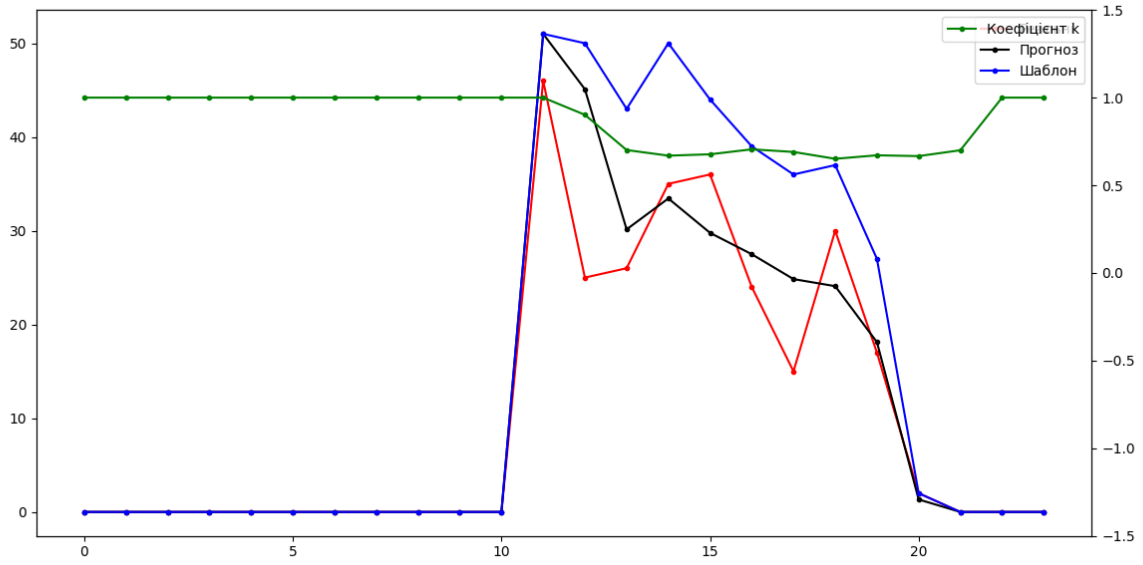

Рис. 2.6. Прогноз

2.3. Пункт 4

Пропоную обрати в якості критерія якості моделі сумарну абсолютну різницю між прогнозом та реальними данними. Це допоможе аналізувати на скільки сумарно прогноз відхилився від реальних данних.

$$\sum_{i=0}^{n}(|\hat{x_i} - x_i|) \tag{1}$$

Де $\hat{x_i}$ - значення прогнозу для години i. А x_i - значення реальних данних для години i.

2.4. Пункт 5

Подивимося як вплине на результативність моделі варіювання обсягу передісторії для обчислення шаблону. Для цього побудуємо графік де зобразимо залежність результатів функції (1) та обсягу передісторії. Та оримаємо, що краще буде обчислювати шаблон на всіх доступних данних.

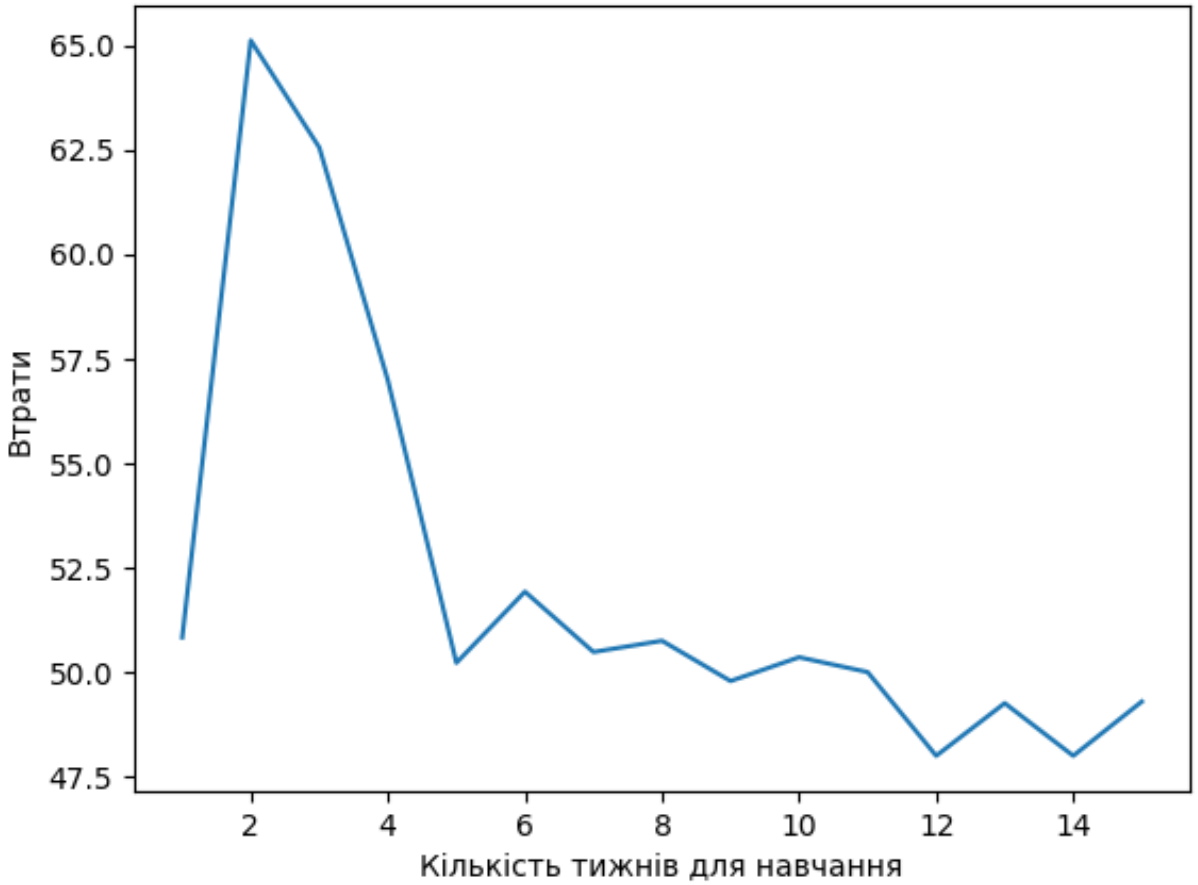

Рис. 2.7. Приклад

2.5. Пункт 6

Подивимося як вплине на результативність моделі варіювання обсягу передісторії для обчислення коефіцієнта. Для цього побудуемо графік де зобразимо залежність результатів функції (1) та обсягу передісторії. Та оримаємо, що краще буде обчислювати коефіцієнт тільки на данних поточного дня для якого й робиться прогноз.

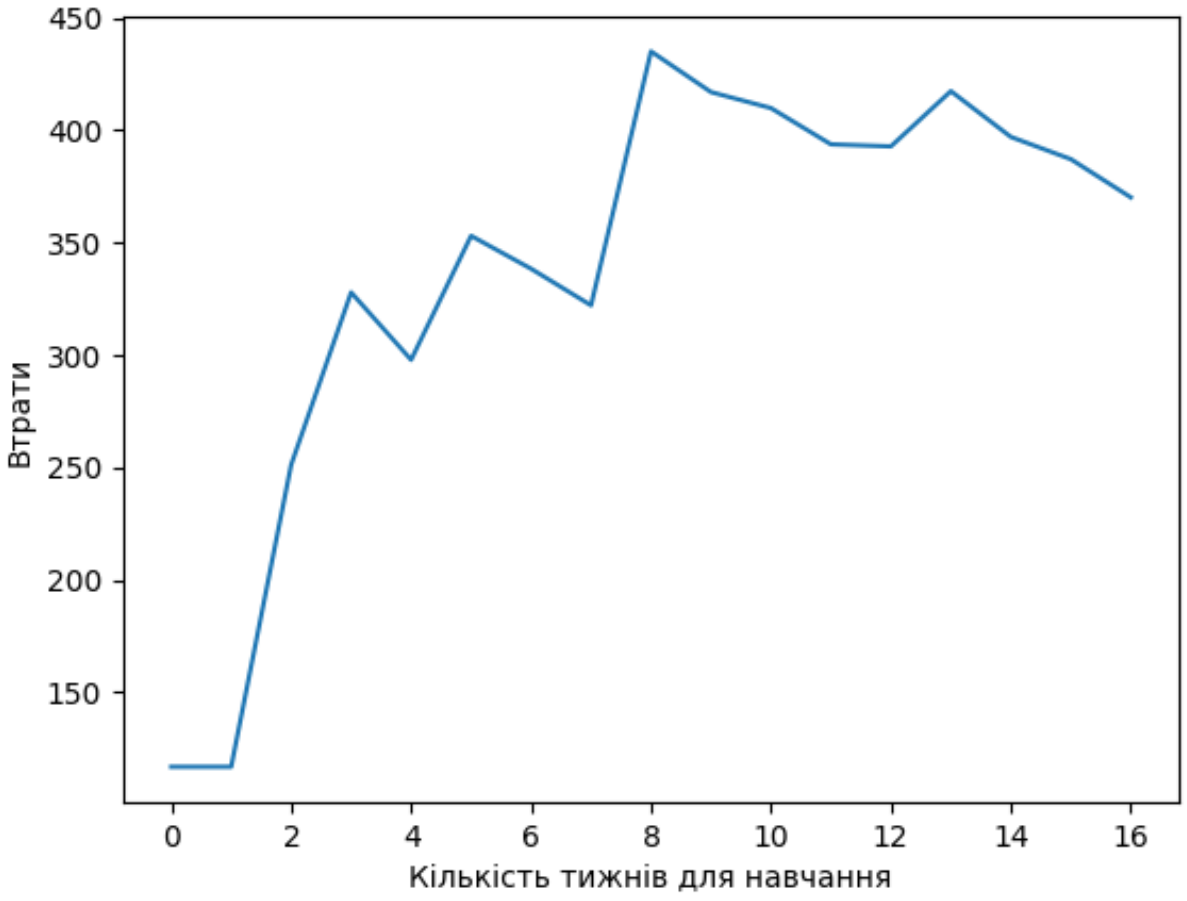

Рис. 2.8. Приклад

2.6. Пункт 7

Подивимося як вплине на результативність моделі, якщо будемо обчислювати шаблон адаптивно. Тобто для поточної години для якої хочемо зробити прогноз будемо проводити аналіз усіх наявніх шабловів(Шаблонів по кожному дню). Для цього для поточної години та переісторії поточного дня будемо рахувати при якому шаблонї функція (1) буде давати мінімальнє значення. Тоді за допомогою шаблона, який показав себе найкраще на предісторії і будемо рахувати наше пе-

редбачення. Якщо передісторія складається лише з нульових значень, то будемо обирати шаблон поточного дня тижня. Подивимося як ця оптимізація буде виглядати графічно.

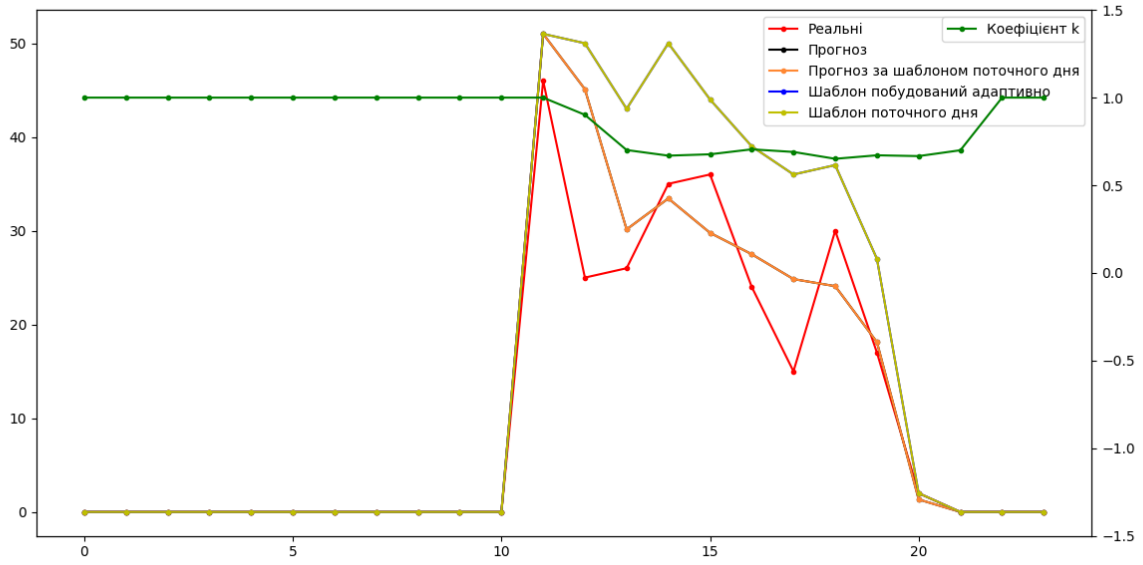

Рис. 2.9. Приклад

Як можна помітити, що шаблон для поточного дня виявився оптимальним тому шаблони співпали. Тому розглянемо ще один приклад. Побудуємо шаблони на всіх данних окрім двох останніх тижнів. А прогноз будемо робити для п'ятниці передостаннього тижня.

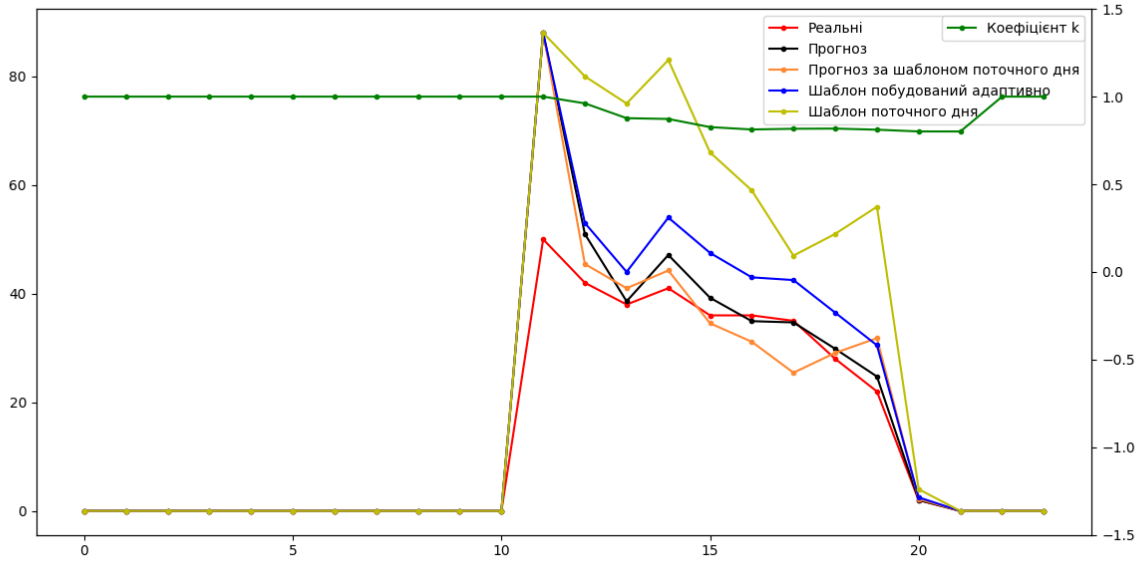

Рис. 2.10. Приклад

У цьому випадку видно, що прогноз на адаптивному шаблоні кращій за звичайний. Але враховуючи випадкову природу значень прогноз за допомогою адаптивного шаблону може бути і гірший ніж за допомогою звичайного шаблону. Для кращого порівняння результатів цих шаблонів треба мати більшу передісторію для магазину, щоб сказати однозначно яких метод буде краще.

2.7. Пункти 8-9

При дослідженні моделі було отримано такі результати. Оптимальним буде використовувати всю наявну інформацію для побудови шаблонів. Для розрахунку коефіцієнта буде краще використовувати тільки передісторію поточного дня. А для обрання методу шаблону треба провести більш глибоке дослідження на більшій кількості данних. Тобто для оптимального розрахунку прогнозу треба максимально збільшити об'єм вхідних данних.

Глава 3

Висновок

У ході лабораторної роботи було побудована математична модель на основі шаблонів та оптимізовано модель за різними критеріями

Глава 4

Код програми

4.1. main.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics import tsaplots
from datetime import datetime, timedelta
import math


class PredictionModel:
    def __init__(self):
        self._template = np.zeros((7, 24))
        self._shop_id = -1
        self._data = None

        self.week_day_by_int = {
            0: "                    ",
            1: "                   ",
            2: "              ",
            3: "              ",
            4: "   ’             ",
            5: "              ",
            6: "              ",
        }

    def count_by_day_and_hour(self):
        ...

    def _count_average_template(self):
        average = lambda sales: sales.mean()
        self._count_template_by_sales(average)

    def _count_median_template(self):
        average = lambda sales: sales.median()
```

```
34          self._count_template_by_sales(average)
35
36      def _count_template_by_sales(self, func):
37          for i, day in enumerate(range(0, 7)):
38              all_week_day_data = self._data[self._data['
                    week_day'] == day]
39              hours = all_week_day_data['time'].unique()
40
41              for j, hour in enumerate(hours):
42                  all_data_in_hour = all_week_day_data[self.
                        _data['time'] == hour]
43                  sales = all_data_in_hour['All']
44
45                  self._template[i][j] = func(sales)
46
47      def _preparing_data_frame(self):
48          self._data["All"] = self._data["All"].replace(np.
                nan, 0)
49          weeks_day = []
50          weeks_id = []
51          week_id = 0
52          last_week_day = 0
53          for date in self._data['date']:
54              date = datetime.strptime(date, "%Y-%m-%d").
                    date()
55              weeks_day.append(date.weekday())
56              if date.weekday() == 0 and last_week_day == 6:
57                  week_id += 1
58              weeks_id.append(week_id)
59              last_week_day = date.weekday()
60
61          self._data['week_day'] = weeks_day
62          self._data['week_id'] = weeks_id
63
64      def _training_by_type(self, training_type):
65          if training_type == 'median':
66              self._count_median_template()
67          elif training_type == 'average':
68              self._count_average_template()
69
```

```
70      def training_model(self, data_frame, training_type='
          median'):
71          self._data = data_frame
72          self._preparing_data_frame()
73          self._training_by_type(training_type)
74
75      def _get_week_by_date(self, date):
76          week = [date]
77
78          date_now = date
79          while True:
80              # print(date_now, date_now.weekday())
81              date_now = date_now + timedelta(days=1)
82              if date_now.weekday() == 0:
83                  break
84
85              week.append(date_now)
86
87          date_now = date
88          while True:
89              # print(date_now, date_now.weekday())
90              date_now = date_now - timedelta(days=1)
91              if date_now.weekday() == 6:
92                  break
93
94              week.insert(0, date_now)
95
96          # print(date)
97          # print(week)
98          return week
99
100     def plot_statistic(self, ploted_days='__all__'):
101         if ploted_days == '__all__':
102             ploted_days = range(0, 7)
103
104         w = math.ceil(math.sqrt(len(ploted_days)))
105         h = math.ceil(len(ploted_days) / w)
106
107         plt.subplots_adjust(wspace=0.3, hspace=0.6)
108         axes = []
109
```

```
110         for i, day in enumerate(ploted_days):
111             ax = plt.subplot(h, w, i+1)
112             axes.append(ax)
113             ax.set_title(self.week_day_by_int[day])
114             ax.set_xlabel('             ')
115             ax.set_ylabel('               ')
116 
117         for i, day in enumerate(ploted_days):
118             all_week_day_data = self._data[self._data['
                week_day'] == day]
119             weeks_dates = all_week_day_data['date'].unique
                ()
120             for week_date in weeks_dates:
121                 week_data = all_week_day_data[self._data['
                    date'] == week_date]
122 
123                 sales = week_data['All']
124                 hours = week_data['time']
125                 axes[i].plot(hours, sales, color='black')
126 
127         reserve_template = self._template.copy()
128 
129         self._training_by_type('median')
130         for i, day in enumerate(ploted_days):
131             sales = self._template[day]
132             hours = range(0, 24)
133 
134             axes[i].plot(hours, sales, color='g', label='
                median')
135 
136         self._training_by_type('average')
137         for i, day in enumerate(ploted_days):
138             sales = self._template[day]
139             hours = range(0, 24)
140 
141             axes[i].plot(hours, sales, color='r', label='
                average')
142 
143         self._template = reserve_template
144 
145         plt.legend()
```

```
146         plt.show()
147
148     def _count_k(self, week_day, data):
149         if len(data) == 0:
150             return 1
151         if data.iloc[-1]['All'] == 0:
152             return 1
153
154         n = 0
155         result = 0
156         for hour_data_id in range(len(data)):
157             hour_data = data.iloc[hour_data_id]
158             if self._template[week_day][hour_data['time']]
                    == 0:
159                 continue
160
161             n += 1
162             result += hour_data['All'] / self._template[
                  week_day][hour_data['time']]
163
164         if n == 0:
165             return 1
166         return result / n
167
168     def predict(self, week_day, hour, history, sub_history
          =None, is_adapt_template=False, template_id=None):
169         if hour == 0:
170             if template_id is not None:
171                 template_id.append(week_day)
172             return 0
173         if sub_history is None:
174             k = self._count_k(week_day, history)
175         else:
176             k = self._count_k(week_day, pd.concat([
                  sub_history, history]))
177
178         if is_adapt_template:
179
180             sales = []
181             for j in range(len(history)):
182                 t_h = history.iloc[:j]
```

```
183                 sales.append(self.predict(week_day, j, t_h
                        , sub_history))
184
185             min_val = self.count_lost(list(history['All'])
                    , sales)
186             t_id = week_day
187             for i in range(self._template.shape[0]):
188                 sales = []
189                 for j in range(len(history)):
190                     t_h = history.iloc[:j]
191                     sales.append(self.predict(i, j, t_h,
                            sub_history))
192
193                 res = self.count_lost(list(history['All'])
                        , sales)
194                 # if hour < 12:
195                 #     print(f'{history["All"]=}')
196                 #     print(f'{sales=}')
197                 #     print(f'{res=}')
198                 #     print(f'{min_val=}')
199                 #     print(f'{t_id=}')
200
201                 # print('='*20)
202                 # print(f'{i=}')
203                 # print(f'{res=}')
204                 # print(f'{min_val=}')
205                 # print(f'{t_id=}')
206                 # print('='*20)
207                 if min_val > res:
208                     min_val = res
209                     t_id = i
210
211             return self.predict(t_id, hour, history,
                    sub_history, template_id=template_id)
212         else:
213             if template_id is not None:
214                 template_id.append(week_day)
215             template = self._template[week_day][hour]
216             return k * template
217
218     def count_lost(self, real, predicted):
```

```
219         assert len(real) == len(predicted)
220
221         lost = 0
222         for i in range(len(real)):
223             lost += abs(real[i] - predicted[i])
224
225         return lost
226
227
228 def set_title(title: str):
229     plt.get_current_fig_manager().set_window_title(title)
230
231
232 def plot_traj_dynamic(df):
233     sales = []
234     dates = df['date'].unique()
235     dates_p = []
236     for date in dates:
237         data_date = df[df['date'] == date]
238         sales.append(sum(data_date['All']))
239         dates_p.append(datetime.strptime(date, "%Y-%m-%d")
              .date())
240
241     plt.xlabel('      ')
242     plt.ylabel('                ')
243     plt.bar(dates_p, sales, color='black')
244     plt.show()
245
246
247 def plot_flow_of_customers(df):
248     sales = df['All']
249     hours = df['time']
250     plt.xlabel('             ')
251     plt.ylabel('               ')
252     plt.scatter(hours, sales)
253     plt.show()
254
255
256 def plot_weekly_flow_of_customers(df):
257     weeks = df['week_id'].unique()
258     for week in weeks:
```

```
259             week_data = df[df['week_id'] == week]
260             hours = week_data['time'] + 24 * week_data['
                  week_day']
261             sales = week_data['All']
262             plt.plot(hours, sales, color='black', linewidth
                  =0.7)
263
264     plt.show()
265
266
267 def plot_autocorrelation(df):
268     x = df['All']
269     fig = tsaplots.plot_acf(x, lags=35)
270     plt.show()
271
272
273 def test_predict(df):
274     df = df.copy()
275     model = PredictionModel()
276     model.training_model(df)
277
278
279     last_week_id = (df['week_id'].unique()[-1])
280     t_data = df[df['week_id'] != last_week_id]
281     r_data = df[df['week_id'] == last_week_id]
282     # print(t_data)
283     # print(r_data)
284
285
286     model.training_model(t_data)
287     # model.training_model(t_data, 'average')
288     # model.plot_statistic('__all__')
289
290
291     day = 1
292     # print(r_data)
293     target_date = r_data[r_data['week_day'] == day].iloc
          [-1]['date']
294     target_data = r_data[r_data['date'] == target_date]
295
296     # print(f'{target_date=}')
```

```
297     # print(target_data)
298 
299     y = []
300     k = []
301     sales = target_data
302     hours = target_data['time']
303     for t in range(len(sales)):
304         y.append(model.predict(day, t, sales.iloc[:t]))
305         k.append(model._count_k(day, sales.iloc[:t]))
306 
307     ax = plt.subplot(1, 1, 1)
308     ax_sub = ax.twinx()
309     ax_sub.set_ylim(-1.5, 1.5)
310     mark = '.'
311     ax.plot(hours, sales['All'], marker=mark, label='
                    ', color='r')
312     ax.plot(hours, y, marker=mark, label='                ',
            color='black')
313     ax.plot(hours, model._template[day], marker=mark,
          label='              ', color='b')
314     ax_sub.plot(hours, k, marker=mark, label='
                           ␣k', color='g')
315     # print(sales, y)
316     lost = model.count_lost(list(sales['All']), list(y))
317     print(f'lost␣=␣{lost}')
318     set_title(f'date:{target_date}␣␣␣␣week␣day:{model.
          week_day_by_int[day]}')
319 
320     ax.legend()
321     ax_sub.legend()
322     plt.show()
323 
324 
325 def test_predict_for_template(df):
326     df = df.copy()
327     model = PredictionModel()
328     # model.training_model(df, 'average')
329     model.training_model(df)
330 
331 
332     last_week_id = (df['week_id'].unique()[-1])
```

```
333     losts = []
334     for id_temp in range(last_week_id):
335         t_data = df[df['week_id'] <= id_temp]
336         r_data = df[df['week_id'] == last_week_id]
337         # print(t_data)
338         # print(r_data)
339 
340 
341         model.training_model(t_data)
342 
343 
344         day = 1
345         # print(r_data)
346         target_date = r_data[r_data['week_day'] == day].
              iloc[-1]['date']
347         target_data = r_data[r_data['date'] == target_date
              ]
348 
349         # print(f'{target_date=}')
350         # print(target_data)
351 
352         y = []
353         k = []
354         sales = target_data
355         hours = target_data['time']
356         for t in range(len(sales)):
357             y.append(model.predict(day, t, sales.iloc[:t])
                  )
358             k.append(model._count_k(day, sales.iloc[:t]))
359 
360         lost = model.count_lost(list(sales['All']), list(y
              ))
361         losts.append(lost)
362 
363     plt.plot(range(last_week_id), losts)
364     plt.show()
365 
366 
367 def test_predict_for_k(df):
368     df = df.copy()
369     model = PredictionModel()
```

```
370     # model.training_model(df, 'average')
371     model.training_model(df)
372
373
374     last_week_id = (df['week_id'].unique()[-1])
375     t_data = df[df['week_id'] != last_week_id]
376     r_data = df[df['week_id'] == last_week_id]
377     # print(t_data)
378     # print(r_data)
379
380
381     model.training_model(t_data)
382
383
384     day = 1
385     # print(r_data)
386     target_date = r_data[r_data['week_day'] == day].iloc
          [-1]['date']
387     target_data = r_data[r_data['date'] == target_date]
388
389
390     losts = []
391     for id_temp in range(last_week_id):
392         y = []
393         k = []
394         # history_added = df[(df['week_id'] <= id_temp) &
              (df['week_day'] == day)]
395         history_added = df[(df['week_id'] <= id_temp)]
396         sales_traget = target_data['All']
397         hours = target_data['time']
398         for t, value in enumerate(sales_traget):
399             sales = target_data.iloc[:t]
400             # print(f'{history_added=}')
401             # print(f'{sales=}')
402             # print(f'{target_data.iloc[:t+1]=}')
403             y.append(model.predict(day, t, sales,
                  history_added))
404             k.append(model._count_k(day, sales))
405
406         lost = model.count_lost(list(sales_traget), list(y
              ))
```

```
407             losts.append(lost)
408
409         plt.plot(range(last_week_id), losts)
410         plt.show()
411
412
413     def test_predict_adapt(df):
414         df = df.copy()
415         model = PredictionModel()
416         model.training_model(df)
417
418
419         last_week_id = (df['week_id'].unique()[-1])
420         t_data = df[df['week_id'] != last_week_id]
421         r_data = df[df['week_id'] == last_week_id]
422
423         # last_week_id = (df['week_id'].unique()[-2])
424         # t_data = df[df['week_id'] < last_week_id]
425         # r_data = df[df['week_id'] == last_week_id]
426         # print(t_data)
427         # print(r_data)
428
429
430         model.training_model(t_data)
431         # model.training_model(t_data, 'average')
432         # model.plot_statistic('__all__')
433
434
435         day = 1
436         # day = 4
437         # print(r_data)
438         target_date = r_data[r_data['week_day'] == day].iloc
              [-1]['date']
439         target_data = r_data[r_data['date'] == target_date]
440
441         # print(f'{target_date=}')
442         # print(target_data)
443
444         y = []
445         y1 = []
446         k = []
```

```
447     sales = target_data
448     hours = target_data['time']
449     template = []
450     for t in range(len(sales)):
451         t_id = []
452         y.append(model.predict(day, t, sales.iloc[:t],
              is_adapt_template=True, template_id=t_id))
453         y1.append(model.predict(day, t, sales.iloc[:t]))
454         template.append(t_id[0])
455         k.append(model._count_k(t_id[0], sales.iloc[:t]))
456 
457     print(f'templates␣ids:␣{template}')
458     template_prep = [model._template[ti][i] for i, ti in
          enumerate(template)]
459 
460     ax = plt.subplot(1, 1, 1)
461     ax_sub = ax.twinx()
462     ax_sub.set_ylim(-1.5, 1.5)
463     mark = '.'
464     ax.plot(hours, sales['All'], marker=mark, label='
                        ', color='r')
465     ax.plot(hours, y, marker=mark, label='                  ',
            color='black')
466     ax.plot(hours, y1, marker=mark, label='                  ␣
              ␣                  ␣                        ␣       ',
          color='#FF8833')
467     ax.plot(hours, template_prep, marker=mark, label='
                      ␣                          ␣
                            ', color='b')
468     ax.plot(hours, model._template[day], marker=mark,
          label='                ␣                      ␣       ',
          color='y')
469     ax_sub.plot(hours, k, marker=mark, label='
                           ␣k', color='g')
470     # print(sales, y)
471     lost = model.count_lost(list(sales['All']), list(y))
472     lost1 = model.count_lost(list(sales['All']), list(y1))
473     print(f'lost␣adapt␣=␣{lost}')
474     print(f'lost␣=␣{lost1}')
475     set_title(f'date:{target_date};␣\tweek␣day:{model.
          week_day_by_int[day]}')
```

```
476
477     ax.legend()
478     ax_sub.legend()
479     plt.show()
480
481
482 def main():
483     df = pd.read_csv('/home/blackgolyb/Documents/labs_opt
          /6/data.csv', sep=';')
484     df = pd.read_csv('data.csv', sep=';')
485     df = df.set_index('index')
486     df_res = df.copy()
487
488     model = PredictionModel()
489     # model.training_model(df, 'average')
490     model.training_model(df)
491
492     plot_traj_dynamic(df)
493     plot_flow_of_customers(df)
494     plot_weekly_flow_of_customers(df)
495     plot_autocorrelation(df)
496     model.plot_statistic('__all__')
497
498     test_predict(df_res)
499     test_predict_for_template(df_res)
500     test_predict_for_k(df_res)
501     test_predict_adapt(df_res)
502
503
504
505 if __name__ == '__main__':
506     main()
```