**Отчёт**

**Алгоритм прогнозирования**

В качестве предсказания результатов в данном алгоритме будет использоваться нейронная сеть. Сама нейронная сеть будет использоваться из framework scikit-learn ­– MLPRegressor.

Для предсказания будущего результата требуется пройти несколько этапов: 1 этап – установка программного продукта, 2 этап – создание проекта, 3 этап – подключение дополнительных библиотек, 4 этап – считывание и обработка БД, 5 этап – работа с MLPRegressor.

**1 ЭТАП – УСТАНОВКА ПРОГРАММНОГО ПРОДУКТА**

Первым делом требуется подготовить нашу машину, на которой будем проводить предсказание результата. Сначала устанавливаем язык [Python 3.7](https://www.python.org/downloads/release/python-370/), после установки нам потребуется установить среду разработки [PyCharm](https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows) для данного языка. Также дополнительно устанавливаем [Anaconda](https://www.anaconda.com/distribution/), она будет выступать в качестве базового интерпретатора. (Установка каждого продукта происходит в соответствии вашей системы)

После установки программного обеспечения, нам потребуется его настроить, для дальнейшего использования. Сначала находим Anaconda Prompt (Anaconda3) и запускаем (Рисунок 1.1)

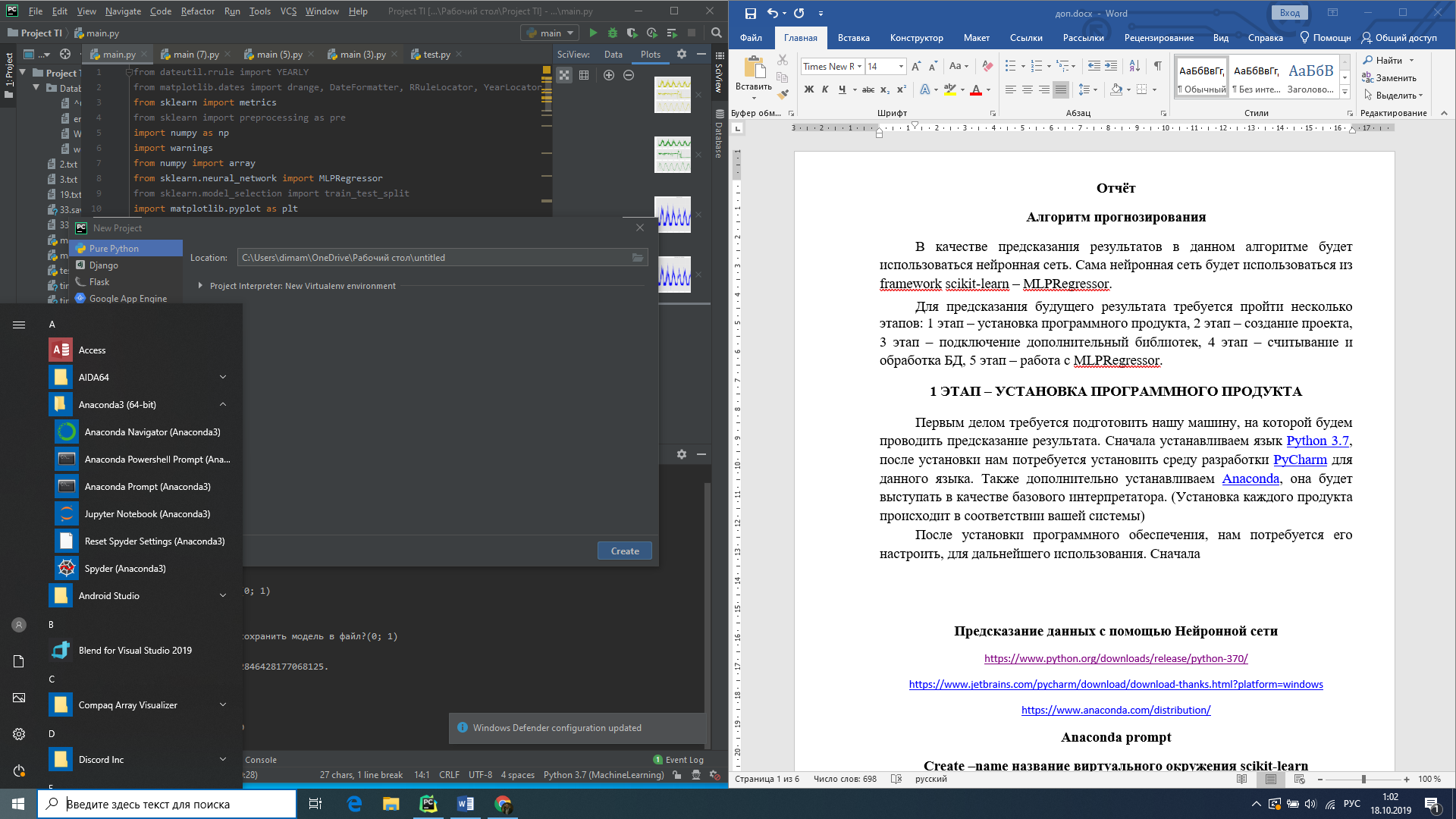


Рисунок 1.1 – Anaconda Prompt (Anaconda3)

После запуска Anaconda Prompt потребуется ввести две строчки:

1. **Create --name *Название\_виртуального\_окружения* scikit-learn** – дополнительно устанавливаем необходимое окружения для нашей машины.
2. **activate *Название\_виртуального\_окружения*** – активируем данное виртуальное окружение.

**2 ЭТАП – СОЗДАНИЕ ПРОЕКТА**

На данном этапе создадим наш проект. Открываем установленный PyCharm и создаем новый проект. При создание нового проекта выбираем Pure Python. (Рисунок 2.1) Также нам потребуется при создании проекта потребуется выбрать новую Виртуальную среду, в качестве базового интерпретатора потребуется выбрать Anacond’у. (Рисунок 2.2) Теперь нам потребуется проверить на работоспособность нашего проекта, для проверки правильности установки всех компонентов можно использовать любой простенький код из интернета.

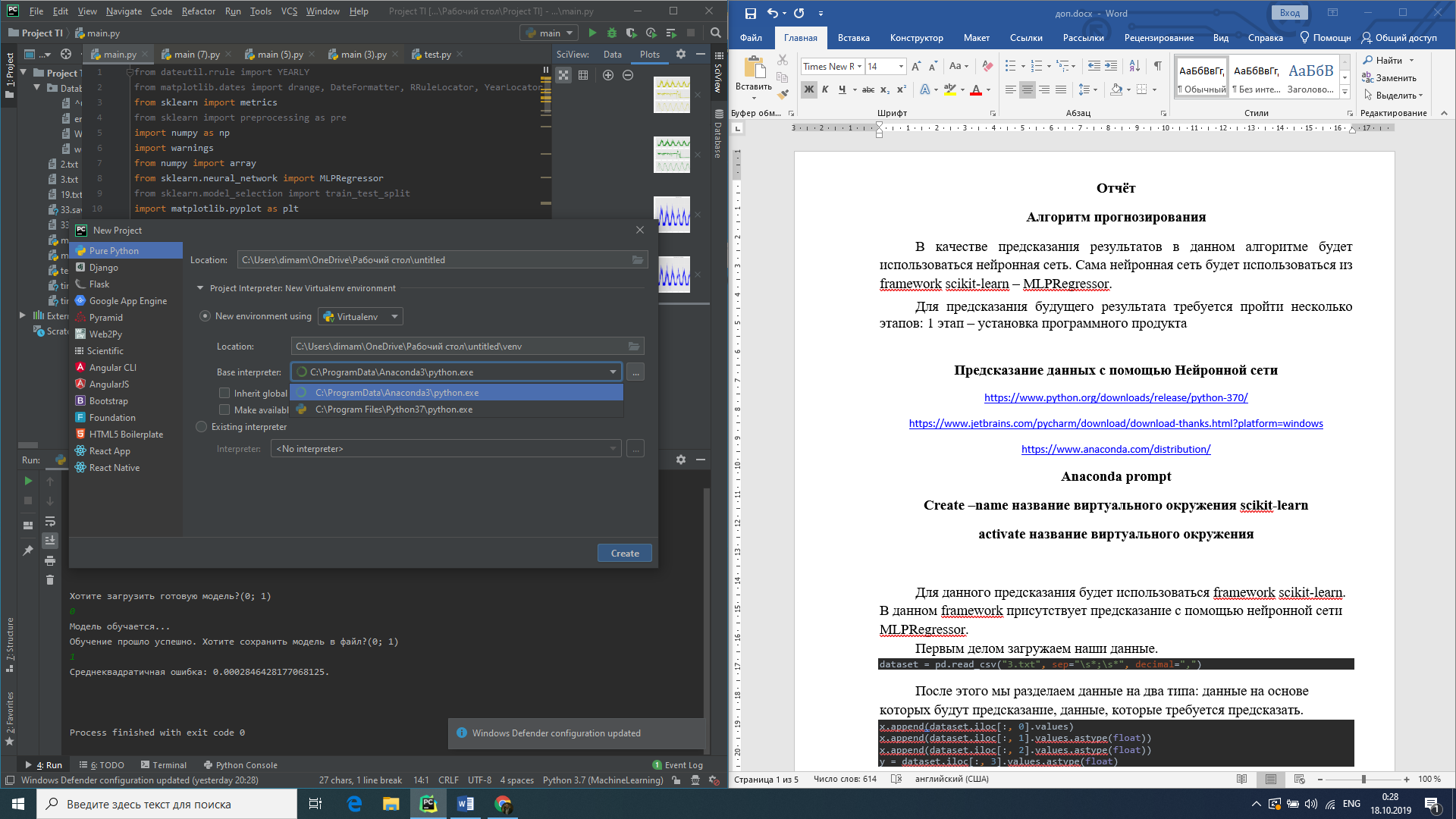
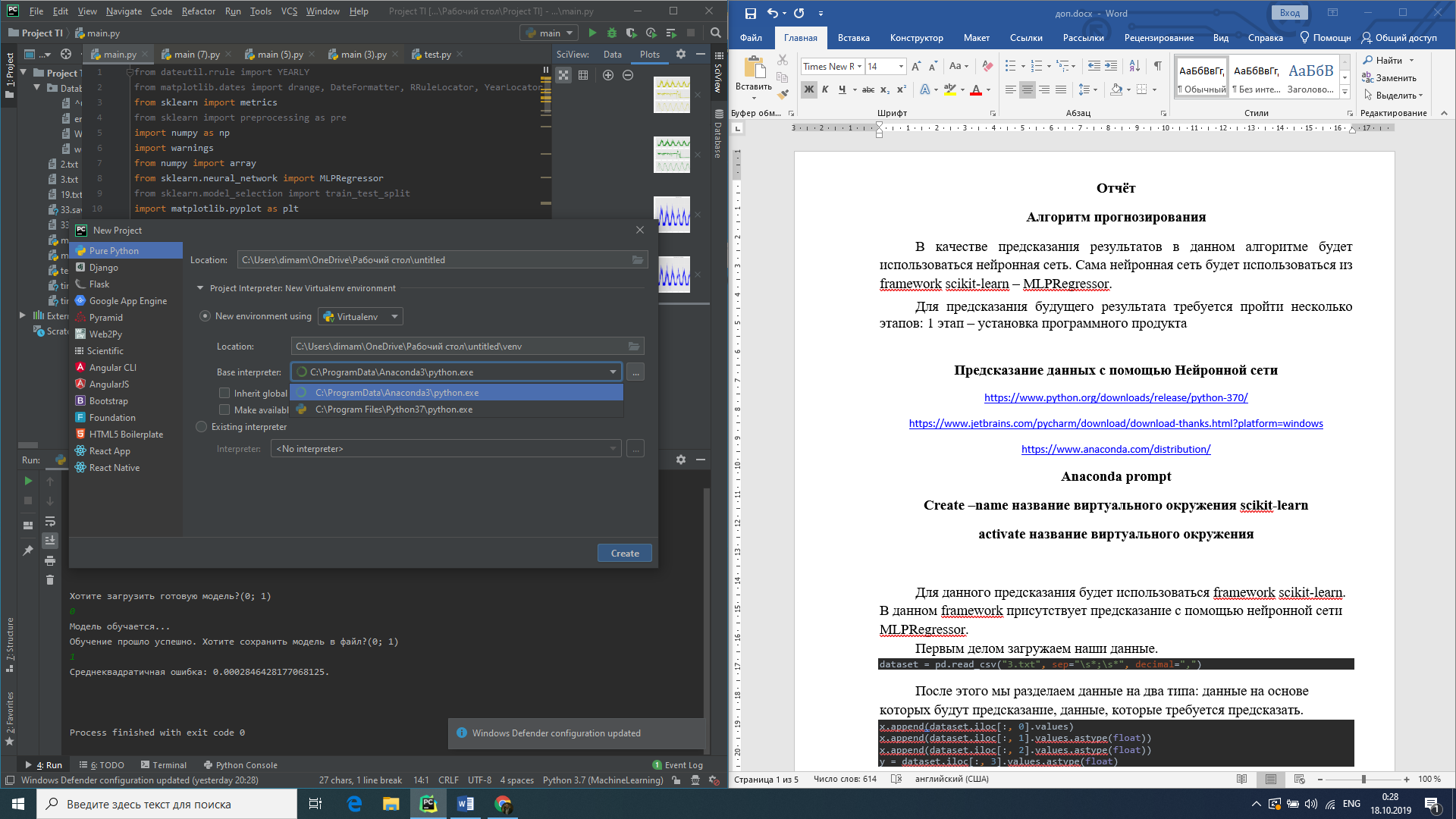


Рисунок 2.1 – Pure Python



1

2

Рисунок 2.2 – Anaconda Prompt (Anaconda3)

**3 ЭТАП – ПОДКЛЮЧЕНИЕ ДОПОЛНИТЕЛЬНЫХ БИБЛИОТЕК**

На данном этапе мы подключаем необходимые библиотеки для нашего проекта. Часть библиотек уже будут встроены, некоторую часть библиотек понадобиться до установить с помощью подсказки или прописав в терминале нашего проекта команду – **pip install *библиотека***. Все необходимые библиотеки будут представлены в листинге 3.1

Листинг 3.1 – импорт библиотек

1. Import copy
2. Import datetime as dt
3. from sklearn import metrics
4. import numpy as np
5. import warnings
6. from numpy import array
7. from sklearn.neural\_network import MLPRegressor
8. import matplotlib.pyplot as plt
9. import pandas as pd
10. import pickle
11. from sklearn.preprocessing import MinMaxScaler

**4 ЭТАП – СЧИТЫВАНИЕ И ОБРАБОТКА БД**

Первоначально нам потребуется считать нашу БД с текстового файла и раскидать наши данные в определенные переменные. В качестве данных предназначенные для помощи в предсказывание мы кидаем в X, а данные часть которых нам потребуется предсказать кидаем в переменную Y. Данная часть кода продемонстрирована в листинге 4.1.

Листинг 4.1 – считывание БД

1. warnings.filterwarnings("ignore")
2. plt.style.use('ggplot')
3. dataset = pd.read\_csv(file\_name + ".txt", sep="\s\*;\s\*", decimal=",")
4. x = list()
5. x.append(dataset.iloc[:, 0].values)
6. x.append(dataset.iloc[:, 2].values.astype(float))
7. x.append(dataset.iloc[:, 3].values.astype(float))
8. y = dataset.iloc[:, 1].values.astype(float)
9. x = array(x)import numpy as np

Проанализировав нашу БД, мы заметим, что данные начинаются в ней с 2012 года и заканчиваются 2019 годом, также в БД каждое измерение происходит каждый час.

В качестве начального значения времени мы берём 1 января 2012 года 00 часов, а в качестве последней даты мы берем 1 января 2019 года 00 часов, в дальнейшем данный промежуток БД будет использоваться для обучения нейронной сети, а записи после 1 января 2019 года 00 часов будут использоваться для предсказания и проверки на точность предсказания. Также мы нормализируем наши данные времени переводя их в секунды, в качестве начального значения 1 января 2012 года 00 часов равная 0 секундам. Выравнивая наши данные времени таким образом, чтобы каждая нацело делилась на 3600. Данный код представлен в листинге 4.2, 4.3 и 4.4.

Листинг 4.2 – Установка начальных значений

1. x\_start = pd.to\_datetime('01.01.2012 00:00:00', errors='raise', format='%d.%m.%Y %H:%M:%S')
2. x\_end = pd.to\_datetime('01.01.2019 00:00:00', errors='raise', format='%d.%m.%Y %H:%M:%S')

Листинг 4.3 – Перевод даты в секунды (начальный отсчёт x\_start)

1. min\_date = x\_start
2. year = array([parser(pd.to\_datetime('01.01.' + str(i) + ' 00:00:00', errors='raise', format='%d.%m.%Y %H:%M:%S')) for i in range(int(x\_start.timetuple()[0]), int(x\_end.timetuple()[0])+2)])
3. x[0] = pd.to\_datetime(x[0], errors='raise', format='%d.%m.%Y %H:%M:%S')
4. max\_date = max(x[0])
5. max\_date = round\_time(max\_date)
6. x[0] = array([parser(item) for item in x[0]])
7. graphics(x, y, year, x\_start, x\_end, 'y-')
8. for i, value in enumerate(x[0]):
9. x[0][i] = (int(round(x[0][i] / 3600))) \* 3600

Листинг 4.4 – перевод времени в секунды

1. def parser(dt):
2. delta = dt - min\_date
3. return int(delta.total\_seconds())

Также для наглядности мы выводим нашу БД на график. (Рисунок 4.1)

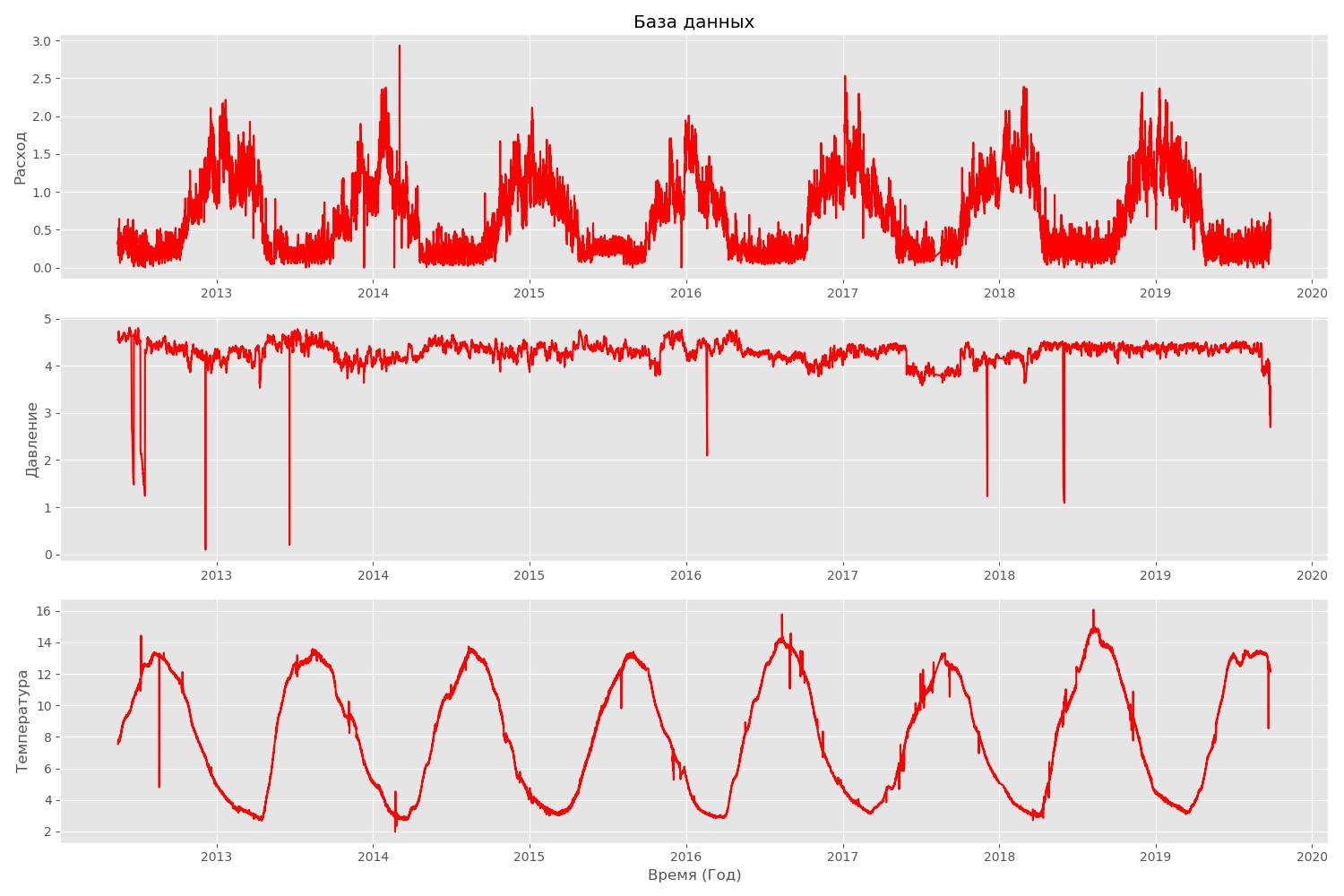


Рисунок 4.1 – график БД

Следующим шагом мы удаляем повторяющиеся данные времени. В том случае, когда T[j] – T[j+1] < -3600, то мы T[j] присваиваем T[j] + 3600 и не удаляем элемент, для сохранения целостности БД. Данный код представлен в листинге 4.5.

Листинг 4.5 – Алгоритм для удаления повторяющихся значений

1. i = 0
2. j = 0
3. tempX0 = x[0]
4. tempX1 = x[1]
5. tempX2 = x[2]
6. tempY0 = y
7. povtoren = 0
8. while i < len(tempX0):
9. if i != (len(tempX0) - 1):
10. j = i + 1
11. while j < len(tempX0):
12. if tempX0[i] == tempX0[j]:
13. if tempX0[i] == tempX0[j]:
14. tempX0 = np.delete(tempX0, j)
15. tempX1 = np.delete(tempX1, j)
16. tempX2 = np.delete(tempX2, j)
17. tempY0 = np.delete(tempY0, j)
18. j = j - 1
19. elif tempX0[j] - tempX0[j + 1] < -3600:
20. tempX0[j] = tempX0[j] + 3600
21. j = j + 1
22. elif tempX0[i] != tempX0[j]:
23. i = j
24. break
25. else:
26. i = i + 1
27. x = list()
28. x.append(tempX0)
29. x.append(tempX1)
30. x.append(tempX2)
31. y = tempY0

Одним из следующих корректировок данных БД является заполнение пропущенных данных, если T[j+1] – T[j] > 3600, то дополняем каждый час пока каждый последующий элемент не будет отличаться на 3600. Данный код представлен в листинге 4.6.

Листинг 4.6 – Интерполяция

1. propusk = 0
2. tempX0 = list()
3. tempX1 = list()
4. tempX2 = list()
5. tempY0 = list()
6. start = 0
7. for i, value in enumerate(x[0]):
8. if i != (len(x[0]) - 1) and x[0][i + 1] - value > 3600:
9. kol = int((x[0][i + 1] - value) / 3600)
10. tempX0.extend(np.ndarray.tolist(x[0][start:i + 1]))
11. tempX1.extend(np.ndarray.tolist(x[1][start:i + 1]))
12. tempX2.extend(np.ndarray.tolist(x[2][start:i + 1]))

Окончание листинга 4.6

1. tempY0.extend(np.ndarray.tolist(y[start:i + 1]))
2. razX1 = (x[1][i + 1] - x[1][i]) / kol
3. razX2 = (x[2][i + 1] - x[2][i]) / kol
4. razY0 = (y[i + 1] - y[i]) / kol
5. for j in range(kol):
6. tempX0.append(x[0][i] + 3600 \* (j + 1))
7. tempX1.append(x[1][i] + razX1 \* (j + 1))
8. tempX2.append(x[2][i] + razX2 \* (j + 1))
9. tempY0.append(y[i] + razY0 \* (j + 1))
10. start = i + 2
11. elif i == (len(x[0]) - 1):
12. tempX0.extend(np.ndarray.tolist(x[0][start:]))
13. tempX1.extend(np.ndarray.tolist(x[1][start:]))
14. tempX2.extend(np.ndarray.tolist(x[2][start:]))
15. tempY0.extend(np.ndarray.tolist(y[start:]))
16. x[0] = array(tempX0)
17. x[1] = array(tempX1)
18. x[2] = array(tempX2)
19. y = array(tempY0)
20. delta = dt - min\_date
21. return int(delta.total\_seconds())

После корректировки БД для наглядности мы выводим нашу БД на график. (Рисунок 4.2)

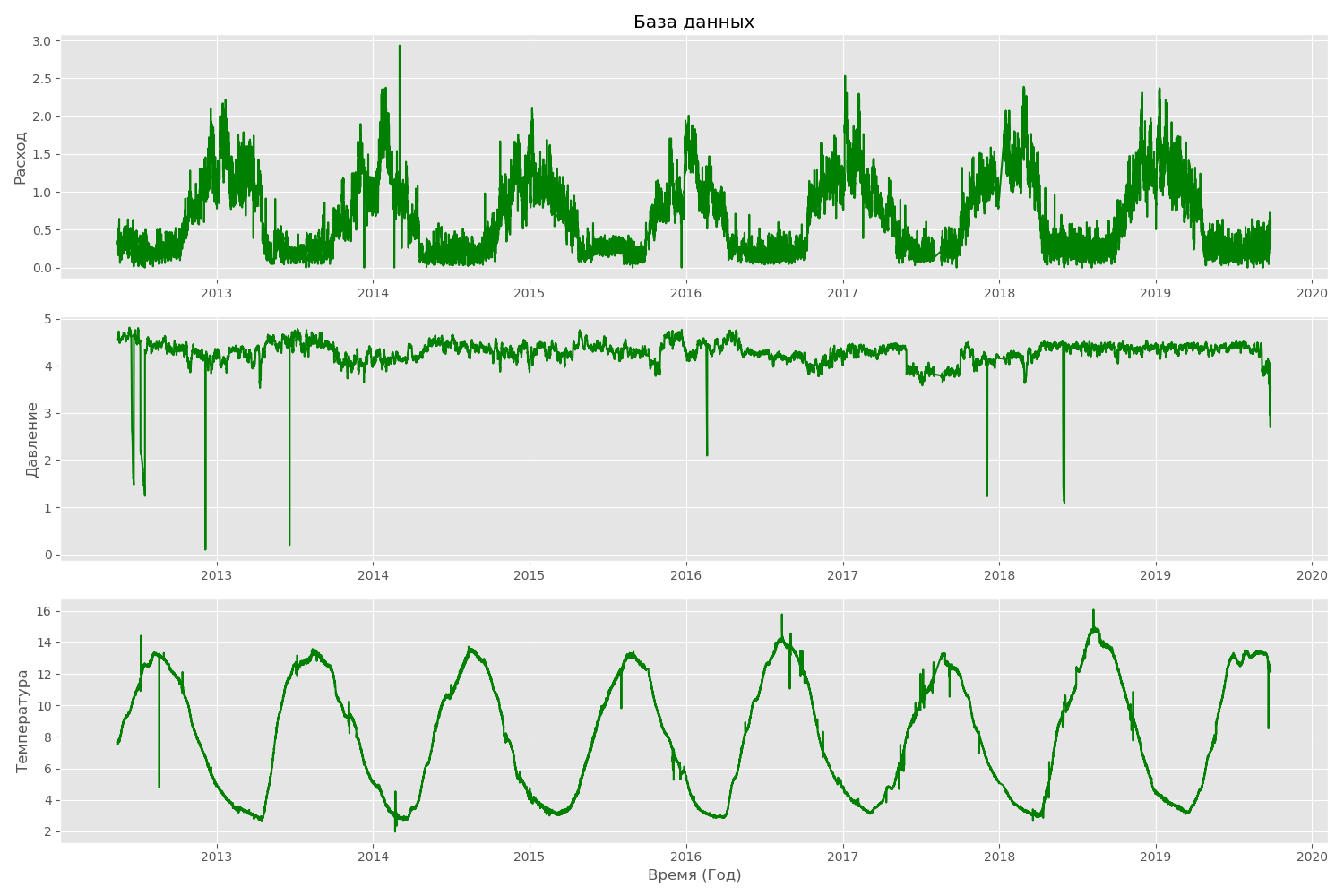


Рисунок 4.1 – график БД

Просмотрев БД, было решено сделать начало каждого года отсчет с нуля. Данное решение приведет к наиболее точному предсказыванию. Данный код с разделением данных представлен в листинге 4.7.

Листинг 4.7 – Проходим по массиву и каждый год начинаем отсчёт с 0

1. number = list(x[0]).index(parser(x\_end))
2. x\_data = copy.deepcopy(x[0])
3. j = 1
4. for i in range(0, len(x[0])):
5. if x[0][i] == years[j]:
6. if j != len(years) - 2:
7. while x[0][i] < years[j + 1]:
8. x[0][i] -= years[j]
9. i += 1
10. j += 1
11. else:
12. while i != len(x[0]):
13. x[0][i] -= years[j]
14. i += 1

Далее мы делим наши данные на две части, первая часть данных x\_train и y\_train будут использоваться для обучения нашей нейронной сети. x\_train – данные в обучающей модели по которым надо будет предсказать y\_train. y\_train – данные, которые потребуется предсказать в обучающей модели с помощью данных x\_train. А также вводим часть данных x\_test и y\_test. y\_test – дынные, которые потребуется уже предсказывать с помощью уже обучившейся Нейронной сети. На вход нашей нейронной сети будут подаваться данные x\_test, на выходе мы получаем данные predicted, которые уже будем сравнивать с данными y\_test. Данный код с разделением данных представлен в листинге 4.8.

Листинг 4.8 – Разделение данных

1. x[1] = array(tempX2)
2. x[2] = array(tempX1)
3. i = list(x[0]).index(parser(x\_end))
4. x\_train = list()
5. x\_test = list()
6. x\_train.append(x[0][:i])
7. x\_train.append(x[1][:i])
8. x\_test.append(x[0][i:])
9. x\_test.append(x[1][i:])
10. y\_train = y[:i]
11. y\_test = y[i:]

Далее мы переходим к следующему этапу, обучение и предсказание.

**5 ЭТАП – РАБОТА С MLPRegressor**

На данном этапе у нас есть два варианта развития:

1. Предсказания расхода с помощью даты. Листинг 5.1.
2. Предсказание расхода с помощью даты и температуры. Листинг 5.2.

Листинг 5.1 – Подача данных времени и расхода

1. plt.figure(figsize=(20, 10))
2. plt.xticks(year, range(int(x\_start.timetuple()[0]), int(x\_end.timetuple()[0])+2))
3. plt.plot(x[0], y, 'b-')
4. regression(x\_data[:number], x\_train[0].reshape(-1, 1), x\_test[0].reshape(-1, 1), y\_train.reshape(-1, 1),
5. y\_test.reshape(-1, 1), MLPRegressor(hidden\_layer\_sizes=(200, 200),
6. solver='lbfgs',
7. #random\_state=9876,
8. activation='relu',
9. max\_iter=1000000000,
10. tol=0.000000001,
11. learning\_rate\_init=0.000001,
12. learning\_rate='adaptive',
13. batch\_size=365\*24\*2,
14. alpha=0,
15. shuffle=True,
16. verbose=True),

Листинг 5.2 – Подача данных времени + температуры и расхода

1. plt.figure(figsize=(20, 10))
2. plt.plot(x[0], y, 'b-')
3. plt.xticks(year, range(int(x\_start.timetuple()[0]), int(x\_end.timetuple()[0]) + 2))
4. regression(x\_data[:number], np.transpose(x\_train), np.transpose(x\_test), y\_train,
5. y\_test, MLPRegressor(hidden\_layer\_sizes=(200, 300, 400),
6. solver='lbfgs',
7. #random\_state=9876,
8. activation='relu',
9. max\_iter=1000000000,
10. tol=0.000000000001,
11. learning\_rate\_init=0.000000000001,
12. learning\_rate='adaptive',
13. batch\_size=365\*24/2,
14. alpha=0,
15. shuffle=True,
16. verbose=True),

Подробно разберем каждый параметр модели. hidden\_layer\_sizes=(100, 100) – количество скрытых слоёв указывается через запятую, а также число нейронов в каждом скрытом слое. solver – решатель для оптимизации весов. activation – активационная функция. max\_iter – максимальное количество обучений нейронной сети. tol ­– допуск для оптимизации, когда потеря или оценка не улучшаются, считается, что конвергенция достигнута, и обучение прекращается. learning\_rate\_init – скорость обучения или размер шага при обновления весов. learning\_rate – скорость обучения. batch\_size – размер минибатчей для стохастических оптимизаторов. alpha – угол активационной функции. shuffle – стоит ли перемешивать образцы в каждой итерации. verbose – вывод каждой обучающей итерации.

После установки всех этих параметров мы передаем данные в наш метод. При использовании Многослойного Перцептрона требуется нормализировать данные с помощью MinMaxScaler для увеличения скорости обучения. После мы подаем данные на обучения или загружаем с файла модель уже обученную. Часть метода представлен в листинге 5.3 и 5.4.

Листинг 5.3 – Метод обучения

1. def regression(x\_graph, x\_train\_reg, x\_test\_reg, y\_train\_reg, y\_test\_reg, model\_reg, title, type\_model, years\_reg):
2. print("Хотите загрузить готовую модель?(0; 1)")
3. user\_enter = input()
4. scaler\_x = MinMaxScaler(feature\_range=(0, 1))
5. scaler\_x = scaler\_x.fit(x\_train\_reg)
6. if user\_enter == '1':
7. print("Введит название файла:")
8. file\_name = input()
9. model\_reg = load\_model(type\_model + file\_name + ".sav")
10. else:
11. print("Модель обучается...")
12. model\_reg.fit(scaler\_x.transform(x\_train\_reg), y\_train\_reg)
13. print("Обучение прошло успешно.")

Листинг 5.4 – Загрузка модели обучения

1. def load\_model(file\_name):
2. return pickle.load(open(file\_name + ".sav", 'rb')) plt.plot(x[0], y, 'b-')

Когда обучили/загрузили нашу модель, то мы предсказываем часть данных БД начиная с 1 января 2019 до окончания самой БД. Получив предсказанные значение сравниваем их с данными, которые были в БД. Таким образом получилось, при предсказывании с помощью только даты MSE = 0.030028, R^2 = 0.878928 (Рисунок 5.1 и 5.2), при предсказывании с помощью даты и температуры MSE = 0.041965, R^2 = 0.830801 (Рисунок 5.3 и 5.4). Также для примера было взята несколько БД и произведены обучения модели и предсказания данных. Файл с данными о предсказания приведен в Predicted.xlsx. Данный код представлен в листинге 5.5.

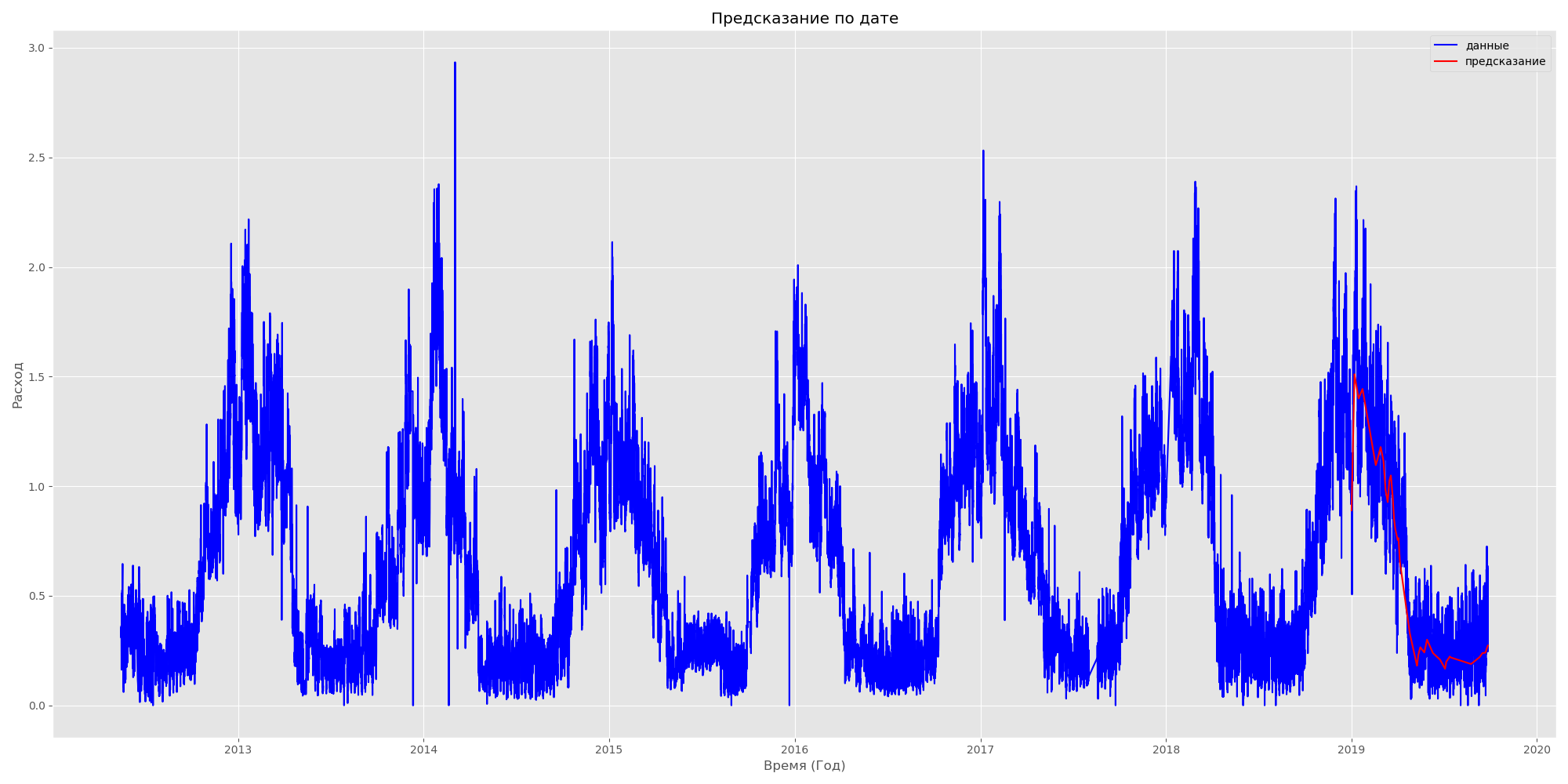


Рисунок 5.1 – график предсказания по времени

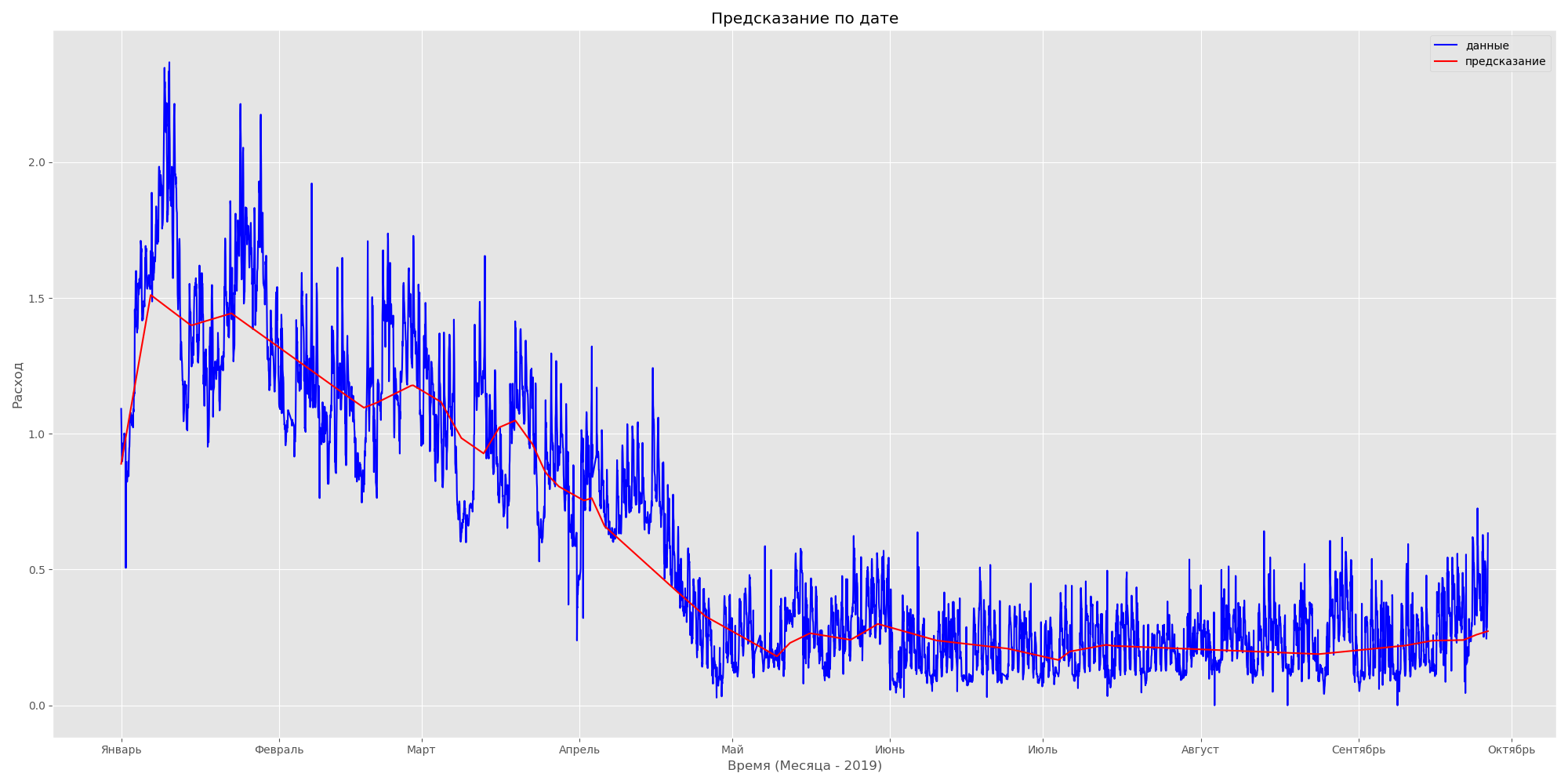


Рисунок 5.2 – график предсказания по времени

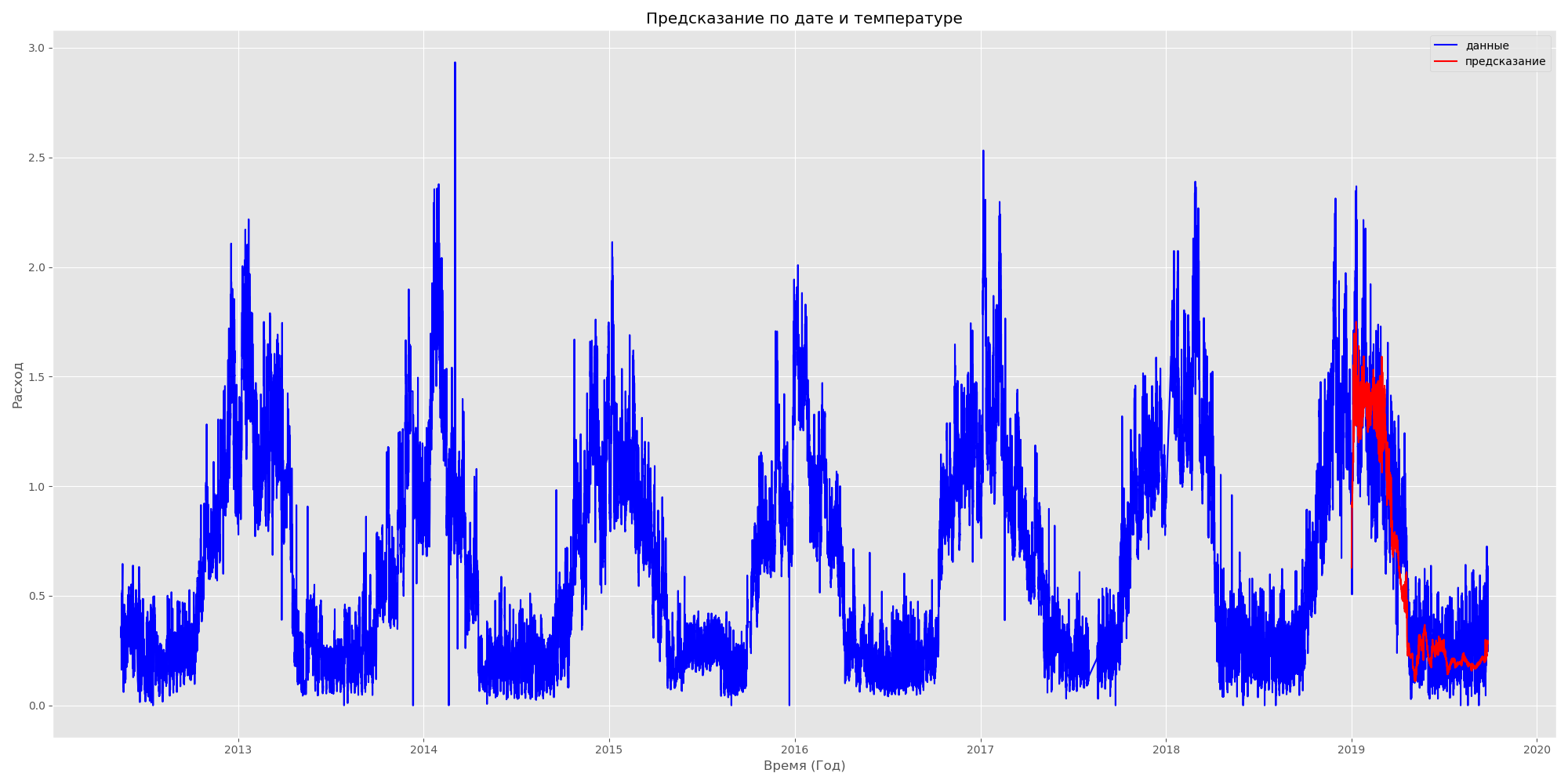


Рисунок 5.3 – график предсказания по времени + температуре

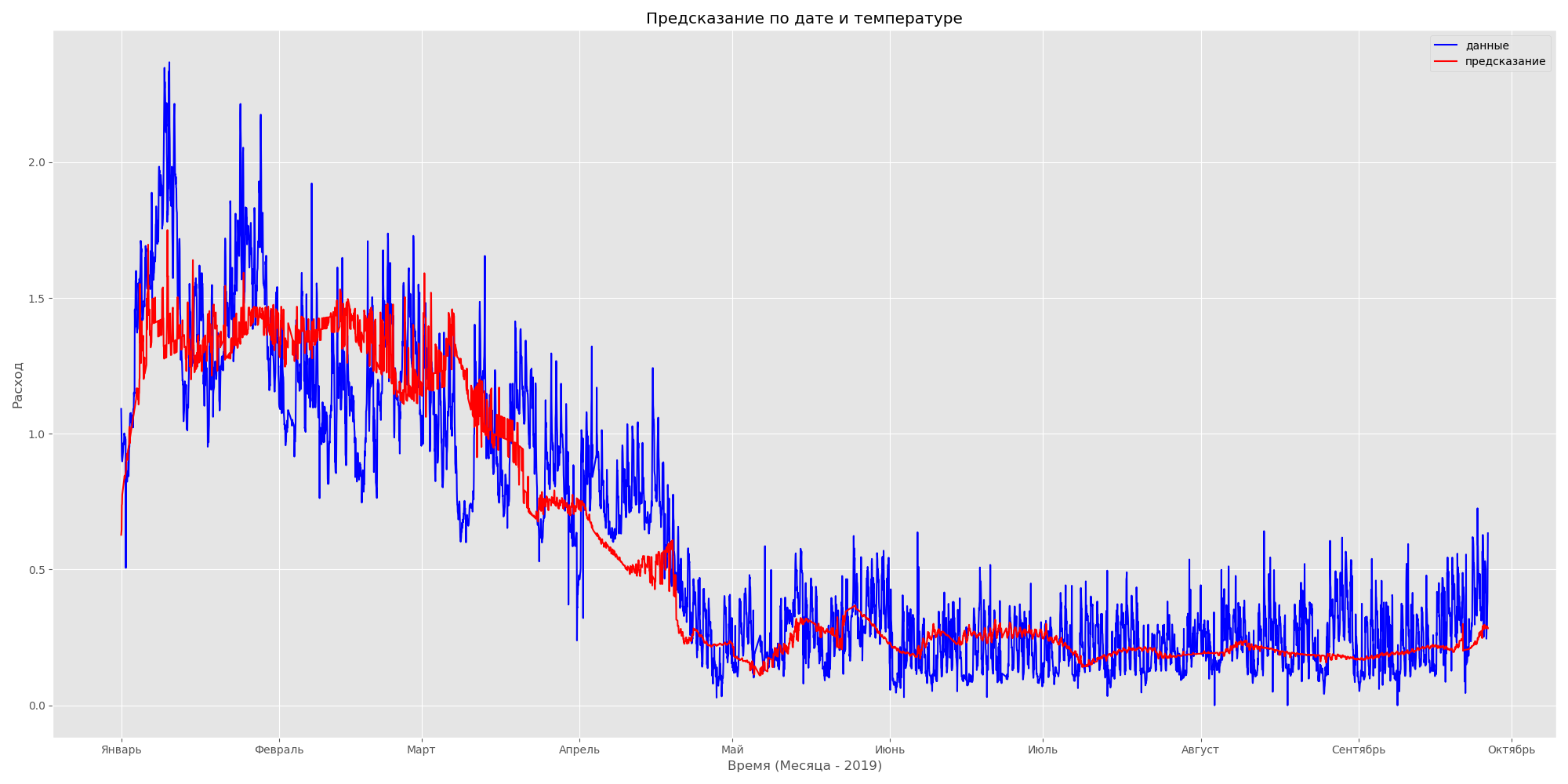


Рисунок 5.4 – график предсказания по времени + температуре

Листинг 5.5 – Предсказание данных и сравнение с частью БД

1. expected\_y = y\_test\_reg
2. model\_reg.out\_activation\_ = 'relu'
3. predicted\_y = model\_reg.predict(scaler\_x.transform(x\_test\_reg))
4. print("Среднеквадратическая ошибка (MSE): {0}.".format(metrics.mean\_squared\_error(expected\_y, predicted\_y)))
5. print("Среднеквадратичная ошибка (R^2): {0}.".format(metrics.r2\_score(expected\_y, predicted\_y))) plt.title(title)
6. plt.plot(x\_graph, predicted\_y, 'r-', label='предсказание')
7. plt.ylabel('Расход')
8. plt.xlabel('Время (Год)')
9. plt.legend(loc='best')
10. plt.show()
11. start\_month = parser(pd.to\_datetime('01.01.2019 00:00:00', errors='raise', format='%d.%m.%Y %H:%M:%S'))
12. months = array([parser(pd.to\_datetime('01.' + str(i) + '.2019 00:00:00', errors= 'raise', format='%d.%m.%Y %H:%M:%S')) - start\_month for i in range(1, 13)])
13. nameMonts = ['Январь', 'Февраль', 'Март', 'Апрель', 'Май', 'Июнь', 'Июль', 'Август', 'Сентябрь', 'Октябрь', 'Ноябрь', 'Декабрь']
14. plt.figure(figsize=(20, 10))
15. plt.title(title)
16. plt.xticks(months, nameMonts)
17. plt.figure(figsize=(20, 10))
18. plt.title(title)
19. plt.xticks(months, nameMonts)
20. if len(np.transpose(x\_test\_reg)) != 1:
21. plt.plot(np.transpose(x\_test\_reg)[0], y\_test\_reg, 'b-', label='данные')
22. plt.plot(np.transpose(x\_test\_reg)[0], predicted\_y, 'r-', label='предсказание')
23. else:
24. plt.plot(x\_test\_reg, y\_test\_reg, 'b-', label='данные')
25. plt.plot(x\_test\_reg, predicted\_y, 'r-', label='предсказание')
26. plt.ylabel('Расход')
27. plt.xlabel('Время (Месяца - 2019)')

Окончание листинга 5.5

1. plt.legend(loc='best')
2. plt.show()

Следующим шагом продлеваем данные до x\_future даты (до 1 января 2021) и предсказываем. Если предсказываем по дате и температуре, то заодно параллельно продлеваем температуру (Рисунок 5.6), а именно предсказываем ее также до x\_future даты (до 1 января 2021) и после подаем данные на предсказывание расхода. Предсказываем расход на будущее с помощью даты (Рисунок 5.5), а также предсказываем расход с помощью даты и температуры (Рисунок 5.7). После всех операции мы можем сохранить модель обучения. Данный код представлен в листинге 5.6, 5.7 и 5.8.

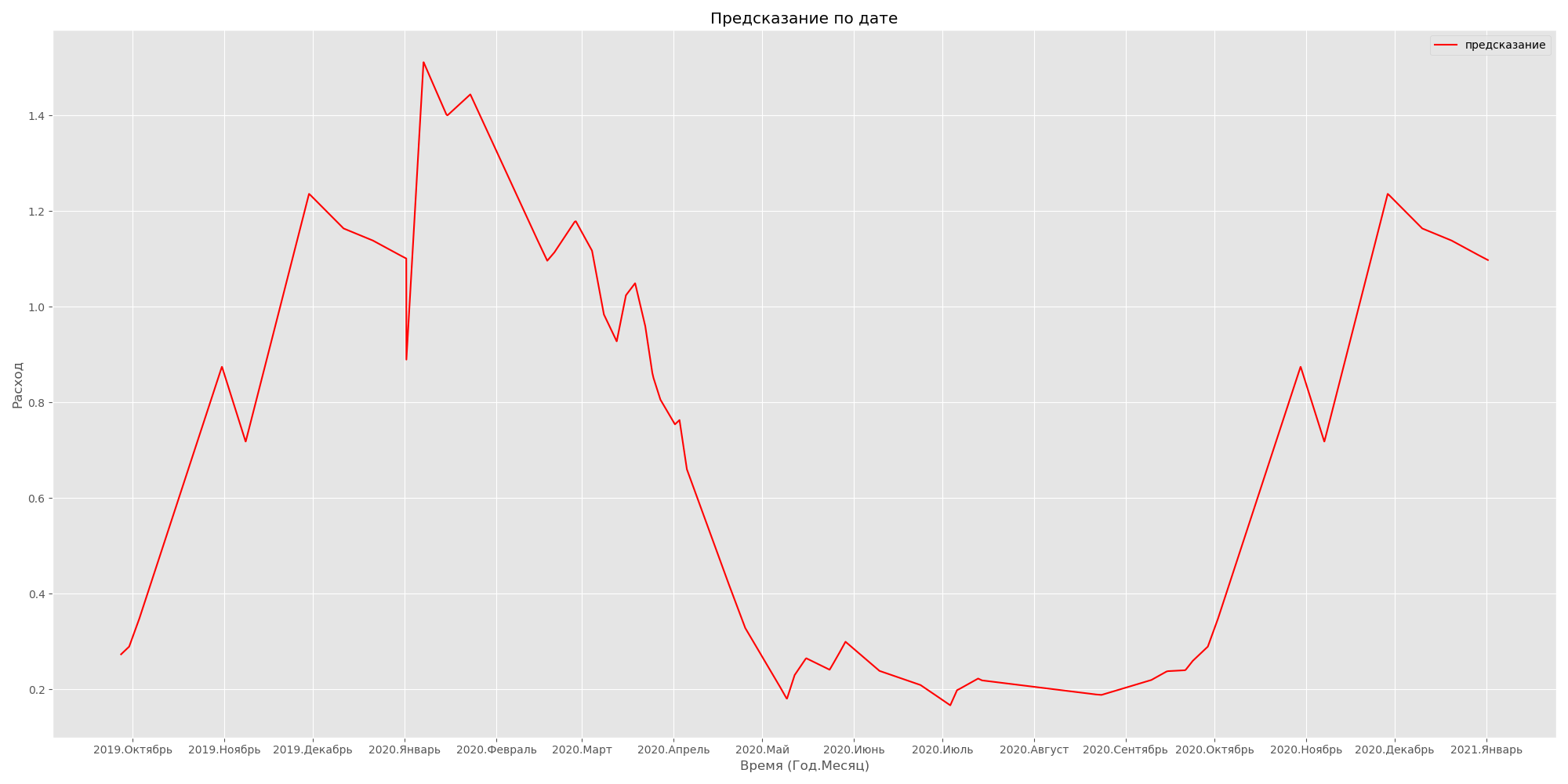


Рисунок 5.5 – график предсказания на будущее по времени

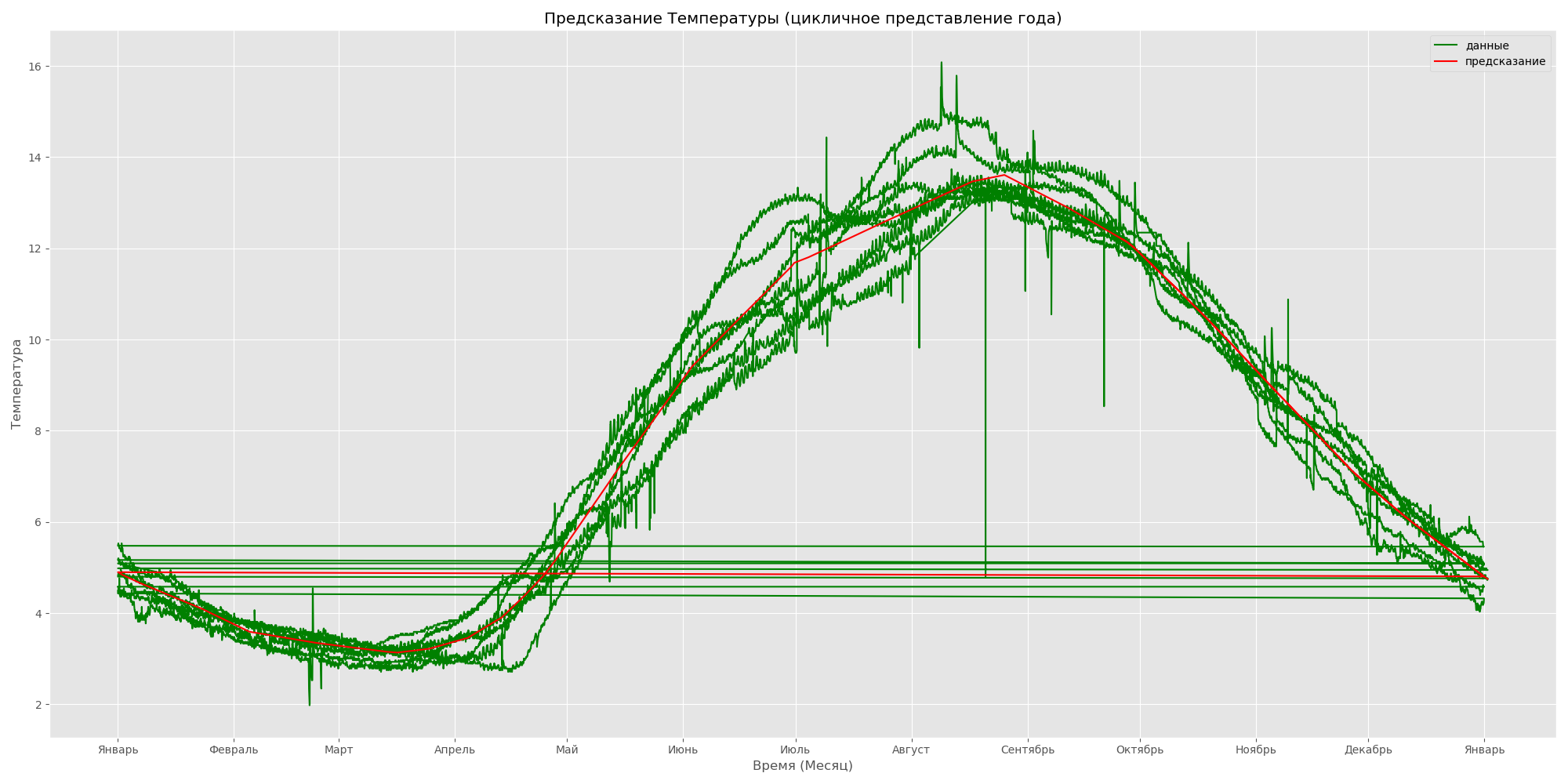


Рисунок 5.6 – график предсказания температуры (цикличное представление года)

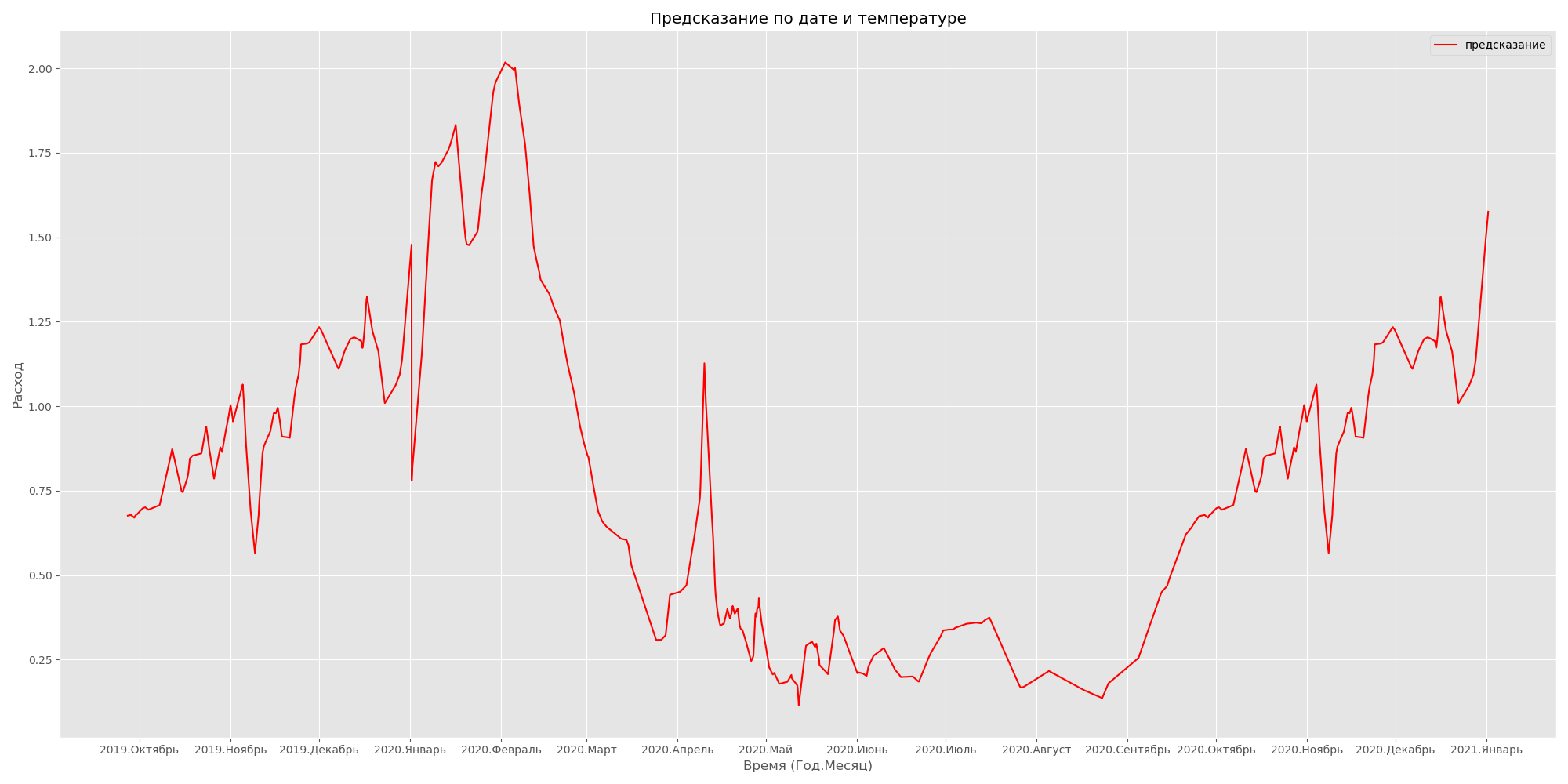


Рисунок 5.7 – график предсказания на будущее по времени и температуре

Листинг 5.6 – Продление данных до x\_future даты и предсказание

1. future\_dates = array(
2. [i for i in range(parser(min\_date), parser(x\_future), 3600)])
3. future\_dates\_periodic = copy.deepcopy(future\_dates)
4. max\_date\_index = list(future\_dates\_periodic).index(parser(max\_date))
5. j = 1
6. for i in range(0, len(future\_dates\_periodic)):
7. if future\_dates\_periodic[i] == years\_reg[j]:
8. if j != len(years) - 2:
9. while future\_dates\_periodic[i] < years\_reg[j + 1]:
10. future\_dates\_periodic[i] -= years\_reg[j]
11. i += 1
12. j += 1
13. else:
14. while i != len(future\_dates\_periodic):
15. future\_dates\_periodic[i] -= years\_reg[j]
16. i += 1
17. future\_dates\_str = [str(i.year) + '.' + str(nameMonts[i.month-1])
18. for i in pd.date\_range(start=str(max\_date.timetuple()[0]) + '.'
19. + str(int(max\_date.timetuple()[1]) + 1),
20. end=str(x\_future.timetuple()[0]) + '.'
21. + str(int(x\_future.timetuple()[1]) + 1), freq='M')]
22. future\_dates\_range = [parser(i)
23. for i in pd.date\_range(start=max\_date, end=x\_future, freq='M')]
24. future\_dates\_periodic = future\_dates\_periodic[max\_date\_index + 1:]
25. if len(np.transpose(x\_test\_reg)) != 1:
26. x\_train\_reg = np.transpose(x\_train\_reg)
27. x\_test\_reg = np.transpose(x\_test\_reg)
28. predicted\_temperatures\_data.append(future\_dates\_periodic)
29. predicted\_temperatures\_data.append(regressionTemperatures(
30. np.concatenate([x\_train\_reg[0], x\_test\_reg[0]]).reshape(-1, 1),

Окончание листинга 5.6

1. np.concatenate([x\_train\_reg[1], x\_test\_reg[1]]),
2. future\_dates\_periodic.reshape(-1, 1), months , nameMonts))
3. scaler\_x2 = MinMaxScaler(feature\_range=(0, 1))
4. predicted\_y = model\_reg.predict(scaler\_x2.fit\_transform(np.transpose( predicted\_temperatures\_data)))
5. else:
6. scaler\_x2 = MinMaxScaler(feature\_range=(0, 1))
7. predicted\_y = model\_reg.predict(scaler\_x2.fit\_transform( future\_dates\_periodic.reshape(-1, 1) ))
8. plt.figure(figsize=(20, 10))
9. plt.title(title)
10. plt.xticks(future\_dates\_range, future\_dates\_str)
11. future\_dates = future\_dates[max\_date\_index + 1:]
12. plt.plot(future\_dates, predicted\_y, 'r-', label='предсказание')
13. plt.ylabel('Расход')
14. plt.xlabel('Время (Год.Месяц)')
15. plt.legend(loc='upper right')
16. plt.show()
17. if user\_enter == '0':
18. print("Хотите сохранить модель в файл?(0; 1)")
19. user\_enter = input()
20. if user\_enter == '1':
21. print("Введит название файла:")
22. file\_name = input()
23. save\_model(model\_reg, type\_model + file\_name + ".sav")

Листинг 5.7 – Предсказание температуры

1. def regressionTemperatures(x\_train\_reg, y\_train\_reg, future\_dates\_periodic, monthsStart, nameMonts):
2. nameMonts.append(nameMonts[0])
3. months = monthsStart.tolist()
4. months.append(monthsStart[11] + monthsStart[1])
5. scaler\_x = MinMaxScaler(feature\_range=(0, 1))
6. scaler\_x = scaler\_x.fit(x\_train\_reg)
7. model = MLPRegressor(hidden\_layer\_sizes=(100, 100),
8. solver='lbfgs',
9. #random\_state=9876,
10. activation='relu',
11. tol=0.0001,
12. learning\_rate\_init=0.001,
13. learning\_rate='adaptive',
14. batch\_size=365 \* 24,
15. alpha=0.001,
16. shuffle=True,
17. verbose=True)
18. model.fit(scaler\_x.transform(x\_train\_reg), y\_train\_reg)
19. predicted\_y = model.predict(scaler\_x.transform(future\_dates\_periodic))
20. plt.figure(figsize=(20, 10))
21. plt.title('Предсказание Температуры (цикличное представление года)')
22. plt.xticks(months, nameMonts)
23. plt.plot(x\_train\_reg, y\_train\_reg, 'g-', label='данные')
24. plt.plot(future\_dates\_periodic, predicted\_y, 'r-', label='предсказание')
25. plt.ylabel('Температура')
26. plt.xlabel('Время (Месяц)')

Окончание листинга 4.6

1. plt.legend(loc='best')
2. plt.show()
3. return predicted\_y

Листинг 5.8 – Сохранение модели обучения

1. def save\_model(model,file\_name):
2. pickle.dump(model, open(file\_name + ".sav", 'wb'))

Разберем метод обучения модели и предсказания данных подробнее. Нам предоставляется возможность загрузки/сохранения модели обучения, для удобства работы. Первым делом мы объявляем нормализатор scaler\_x = MinMaxScaler(), который нам поможет нормализировать данные для более точного предсказания.

Следующим шагом отдаем наши данные на обучения, а также параллельно нормализируем данные:

– model\_reg.fit(scaler\_x.fit\_transform(x\_train\_reg), y\_train\_reg)

Далее мы после обучения нашей нейронной сети, мы передаем данные, по которым нам требуется предсказать расход:

– predicted\_y = model\_reg.predict(scaler\_x.fit\_transform(x\_test\_reg))

Далее в качестве проверки наших предсказаний, используется Средний квадрат отклонения и Среднеквадратичная ошибка (Формулы представлены в Доп.Материале):

– print("Среднеквадратическая ошибка (MSE):

{0}.".format( metrics.mean\_squared\_error( expected\_y, predicted\_y))) – чем ближе к нулю тем лучше

– print("Коэффициент детерминации (R^2):

{0}.".format( metrics.r2\_score(expected\_y, predicted\_y))) – чем ближе к единице тем лучше

Исходный код проекта [здесь](https://github.com/Dimar24/Machine-learning/blob/master/main.py).

**Дополнительный материал**

Коэффициент детерминации (R^2 - R-квадрат) – это доля дисперсии зависимой переменной, объясняемая рассматриваемой моделью. Более точно – это единица минус доля необъяснённой дисперсии (дисперсии случайной ошибки модели, или условной по признакам дисперсии зависимой переменной) в дисперсии зависимой переменной.

R^2 =1-\frac {\hat{\sigma}^2}{\hat{\sigma}^2_y}=1-\frac {RSS/n}{TSS/n}=1-\frac {RSS} {TSS}, где

RSS=\sum^n_{t=1}e^2_t=\sum^n_{t=1} (y_t-\hat y_t)^2

сумма квадратов регрессионных остатков,

TSS=\sum^n_{t=1} (y_t-\bar{y})^2=n \hat \sigma^2_y

общая дисперсия.

Среднеквадратическая ошибка (MSE) – среднеквадратическая ошибка прогноза применяется в ситуациях, когда нам надо подчеркнуть большие ошибки и выбрать модель, которая дает меньше больших ошибок прогноза.

RSS=\sum^n_{t=1}e^2_t=\sum^n_{t=1} (y_t-\hat y_t)^2

сумма квадратов регрессионных остатков,

В нашей Нейронной сети используется Многослойного Перцептрона (MLP – Multi-layer Perceptron) –контролируемый алгоритм обучения, который изучает функцию путем обучения на наборе данных, где количество измерений для ввода и количество измерений для вывода. Учитывая набор функций X = x1, x2, …, xm и цель y, он может выучить аппроксиматор нелинейных функций для классификации или регрессии. Он отличается от логистической регрессии тем, что между входным и выходным слоями может быть один или несколько нелинейных слоев, называемых скрытыми слоями. На рисунке доп.1 показан MLP с одним скрытым слоем и скалярным выводом.

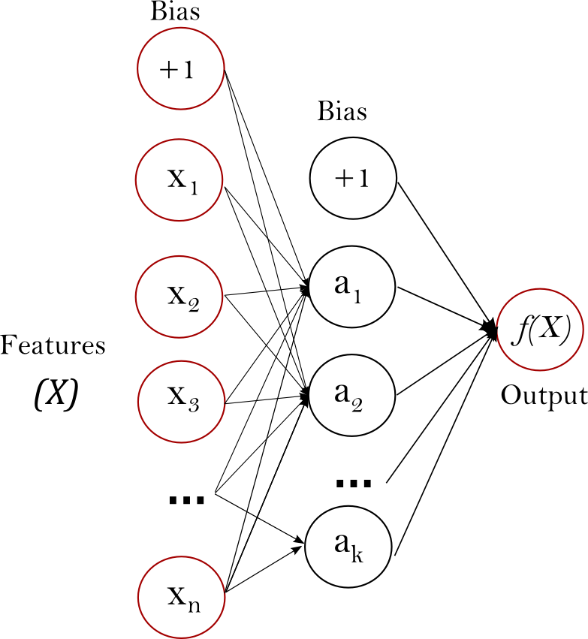


Рисунок доп.1 – Нейронная сеть многослойного персиптрона

Самый левый слой, известный как входной слой, состоит из набора нейронов {xi |x1, x2, …, xm} представляет входные функции. Каждый нейрон в скрытом слое преобразует значения из предыдущего слоя с помощью взвешенного линейного суммирования w1x1+ w2x2+…+ wmxm последующей нелинейной функцией активации - как гиперболическая функция загара. Выходной слой получает значения из последнего скрытого слоя и преобразует их в выходные значения.

Модуль содержит публичные атрибуты coefs\_и intercepts\_. coefs\_список весовых матриц, где весовая матрица по индексу i представляет веса между слоями i и слой i+1, intercepts\_список векторов смещения, где вектор по индексуя представляет значения смещения, добавленные к слою i+1,

Преимущества Многослойного Перцептрона:

* Способность изучать нелинейные модели.
* Возможность изучать модели в режиме реального времени (онлайн обучение) с использованием partial\_fit.

К недостаткам Многослойного перцептрона (MLP) относятся:

* MLP со скрытыми слоями имеют невыпуклую функцию потерь, где существует более одного локального минимума. Поэтому разные инициализации случайного веса могут привести к различной точности проверки.
* MLP требует настройки ряда гиперпараметров, таких как количество скрытых нейронов, слоев и итераций.
* MLP чувствителен к масштабированию объектов.

Далее наглядно представлена сама нейронная сеть.

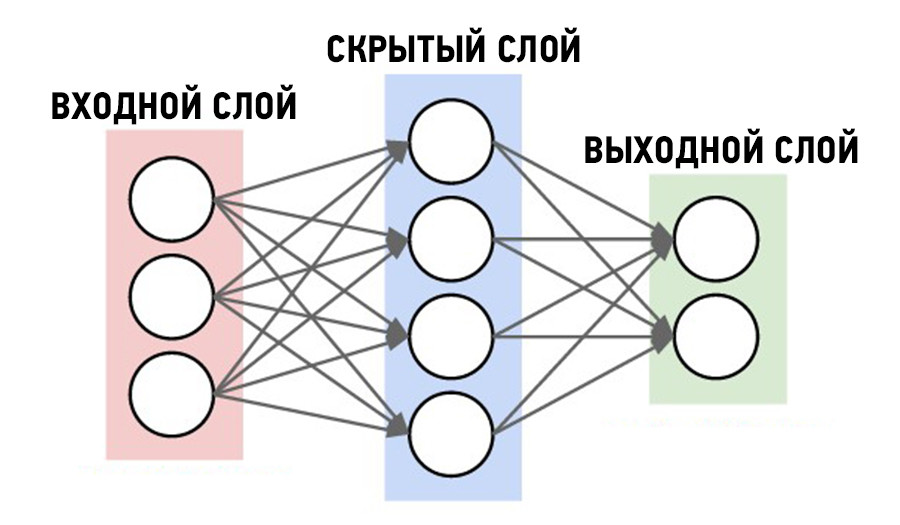


Рисунок доп.2 – Общая картина нейронной сети

**ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ**

1. <https://www.geeksforgeeks.org/ml-sklearn-linear_model-linearregression-in-python/>
2. <https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155>
3. <https://www.machinelearningmastery.ru/persistence-time-series-forecasting-with-python/>
4. <https://programtalk.com/python-examples/sklearn.neural_network.MLPRegressor/>
5. <https://stackoverflow.com/questions/46616883/how-to-set-numbers-of-epoch-in-scikit-learn-mlpregressor>
6. <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html>
7. <https://www.programcreek.com/python/example/93778/sklearn.neural_network.MLPRegressor>
8. <http://qaru.site/questions/17911571/neural-network-epoch-and-maxiter-in-scikit-learn>