

Politechnika Gdańska

Wydział Fizyki Technicznej i Matematyki Stosowanej

Podyplomowe Studia „Inżynieria danych – Data Science”



***Analiza i predykcja zużycia energii elektrycznej w
budynkach mieszkaniowych***

Mateusz Grulkowski

Paweł Ciechanowicz

Sławomir Lisowski

Opiekun pracy dyplomowej: prof. dr hab. Józef E. Sienkiewicz

Gdańsk, 06.2020

Spis treści

Wstęp.....	5
1. Pobranie danych	6
2. Przygotowanie danych do modelowania.....	9
2.1 Etapy przygotowania danych	9
2.1.1 Umieszczenie danych poprzez interfejs Apache Ambari w HDFS (Hadoop Distributed File System).....	9
2.1.2 Dodanie zmiennych ze zbioru metadanych.....	10
2.1.3 Pobranie oraz dołączenie do ramki informacji geolokalizacyjnych i danych pogodowych	11
2.1.4 Pozostałe przekształcenia danych	14
3. EDA - Eksploracyjna Analiza Danych.....	16
3.1 Importowanie potrzebnych bibliotek oraz pakietów	16
3.2 Importowanie danych oraz łączenie w jeden zbiór	16
3.3 Spojrzenie na strukturę danych:	16
3.3.1 Podsumowanie:	17
3.3.2 Wnioski:	18
3.3.3 Spojrzenie na 6 pierwszych wierszy	19
3.4 Analiza korelacji zmiennych liczbowych	20
3.4.1 Wnioski:	20
3.5 Analiza zmiennej ‘grid’ - zużycie prądu	20
3.5.1 Wykres rozkładu zmiennej zużycie prądu	21
3.5.2 Wykres rozkładu zmiennej zużycie prądu z podziałem na stany	22
3.5.3 Wnioski:	22
3.5.4 Wykres pudełkowy prezentujący zużycie prądu w ciągu doby	23
3.5.5 Wykres pudełkowy prezentujący zużycie prądu w ciągu doby w podziale na stany	23

3.5.6 Wnioski:.....	24
3.6 Analiza zmiennej ‘temp_avg’ - średniej temperatury.....	24
3.6.1 Wykres rozkładu średniej temperatury	24
3.6.2 Wykres rozkładu średniej temperatury z podziałem na stany	25
3.6.3 Wnioski:.....	25
3.6.4 Wykres pudełkowy prezentujący średnie temperatury w ciągu doby	26
3.6.5 Wykres pudełkowy prezentujący średnią temperaturę w ciągu doby w podziale na stany	26
3.6.6 Wnioski:.....	27
3.7 Analiza zmiennej ‘humidity_avg’ - średniej wilgotności.....	27
3.7.1 Wykres rozkładu średniej wilgotności.....	27
3.7.2 Wykres rozkładu średniej wilgotności z podziałem na stany	28
3.7.3 Wnioski:.....	28
3.7.4 Wykres pudełkowy prezentujący średnią wilgotność w ciągu doby	29
3.7.4 Wykres pudełkowy prezentujący średnią wilgotność w ciągu doby w podziale na stany	29
3.7.5 Wnioski:.....	30
3.8 Wskazanie zależności między średnią temperaturą a wilgotnością.....	30
3.9 Analiza zmiennej ‘wind_speed_avg’ - średniej prędkości wiatru	30
3.9.1 Wykres rozkładu średniej prędkości wiatru z podziałem na stany	31
3.9.3 Wykres pudełkowy prezentujący średnią prędkość wiatru w ciągu doby w podziale na stany	31
3.9.4 Wnioski:.....	31
3.10 Analiza zmiennej ‘wind_dir_avg’ - kierunku wiatru wyrażonego w stopniach 0-360	32
3.10.1 Wykres rozkładu kierunku wiatru.....	32
3.10.2 Wnioski:.....	32
3.11 Analiza zmiennej ‘pressure_max’ - ciśnienia	33

3.11.1 Wykres rozkładu ciśnienia z podziałem na stany.....	33
3.11.2 Wnioski:	33
3.11.3 Lokalizacje występujące w analizie, na podstawie długości i szerokości geograficznej.	33
3.11.4 Wykresy zależności zużycia prądu od zmiennej ‘holiday’, przedstawiające stosunek ilości obserwacji oraz rozkład	34
3.11.5 Wnioski:	35
4. Inżynieria cech	36
5. Badanie modeli regresji.....	39
6. Wynik i wnioski	41

Wstęp

Celem pracy jest stworzenie modelu pozwalającego na predykcję zużycia energii elektrycznej na podstawie prognozy pogody oraz parametrów budynku. Dane dotyczące budynków oraz zużywanej przez nie energii zostały pobrane ze strony pecanstreet.org, natomiast do pobrania danych pogodowych wykorzystano API serwisu wunderground.com. Wykorzystane w pracy informacje pochodzą z domów znajdujących się w stanach Kalifornia i Teksas. Pomiarzy wykonane w Kalifornii zawierają dane z okresu od 01.01.2014 do 31.12.2014, natomiast dane z Teksasu to okres od 01.01.2018 do 31.12.2018. Dane pomiarowe były zbierane w 15 minutowych odstępach. Historyczne dane pogodowe zawierają: temperaturę, wilgotność powietrza, ciśnienie atmosferyczne, prędkość wiatru oraz kierunek wiatru.

Poszczególnymi etapami pracy są:

- pobranie danych pogodowych z wykorzystaniem API,
- scalenie danych pochodzących z wielu domów w jeden plik formatu csv,
- stworzenie potoku pozwalającego na przetworzenie danych do interesującej nas formy
- eksploracyjna analiza danych,
- inżynieria cech i wybór najbardziej znaczących,
- zbadanie modeli regresji w celu wybrania potencjalnie najlepszego modelu do danego zadania

1. Pobranie danych

Dane dotyczące zużycia energii elektrycznej zostały pobrane ze strony pecanstreet.org. Firma ta zajmuje się opomiarowaniem budynków jedno i wielorodzinnych w zakresie zużycia i generacji prądu, wody, gazu ziemnego. Dane dostępne do celów akademickich zawierają zużycie energii elektrycznej w domach i mieszkaniach we wszystkich obwodach elektrycznych, w tym danego dotyczące produkcji energii elektrycznej przez panele fotowoltaiczne. Historyczne dane pogodowe zostały pobrane przy wykorzystaniu API strony wunderground.com. Serwis ten agreguje dane wysyłane przez użytkowników posiadających zatwierdzone stacje pogodowe. Parametry pogodowe jakie zostały wybrane w tej pracy to: temperatura powietrza, wilgotność powietrza, ciśnienie atmosferyczne, prędkość wiatru i kierunek wiatru. Podczas wykorzystywania API okazało się, że nie wszystkie stacje pogodowe posiadały dane z całego interesującego nas przedziału czasu dla danego domu/mieszkania, co wymusiło przerobienie programu w taki sposób, aby w przypadku braku danych przeszukiwał wszystkie najbliższe stacje pogodowe w celu uzupełnienia brakujących informacji.

Kod, który wykorzystano do pobrania danych za pomocą API:

```
def get_weather_data(date, station_id):
    time.sleep(4)
    frmt = 'json'
    units = 'm'
    api_key = WEATHER_API
    url = 'https://api.weather.com/v2/pws/history/hourly'

    headers = {'Accept-Encoding': 'gzip, deflate, br'}

    payload = {'stationId': station_id, 'format': frmt,
               'units': units, 'date': date, 'apiKey': api_key}
    r = requests.get(url=url, params=payload, headers=headers)

    return r

def get_new_station(date, lat, lng):
    geocode = str(lat) + ',' + str(lng)
    product = 'pws'
    frmt = 'json'
    api_key = WEATHER_API
    payload = {'geocode': geocode, 'product': product,
               'format': frmt, 'apiKey': api_key}

    headers = {'Accept-Encoding': 'gzip, deflate, br'}

    url = 'https://api.weather.com/v3/location/near'
```

```

r = requests.get(url=url, params=payload, headers=headers)
response = r.json()
stations = response['location']['qcStatus']
for index, value in enumerate(stations):
    if value == 1:
        station_id = response['location']['stationId'][index]
        new_station = get_weather_data(date, station_id)
        if new_station.status_code == 200:
            return new_station

return 'None'

def download_weather_data(date, station_id):
    r = get_weather_data(date, station_id)

    if r.status_code == 200:
        result = r.json()
        filepath = "weather_data/" + date + "_" + station_id + ".json"

        with open(filepath, "w+") as outfile:
            json.dump(result, outfile)
    else:
        if r.status_code == 401:
            print("Klucz API został zablokowany")
        else:
            print("Station: " + station_id +
                  " date: " + date + " code:" + str(r.status_code))

    return r.status_code

# Parametry pobierane z API: tempAvg, windspeedAvg, pressureMax,
# humidityAvg, winddirAvg
def get_weather(timestamp, station_id):

    if station_id == 'None':
        return [np.nan, np.nan, np.nan, np.nan, np.nan]

    date = create_date(timestamp)
    path = "weather_data/" + date + "_" + station_id + ".json"

    if not os.path.exists(path):
        download_weather_data(date, station_id)

    try:
        with open(path, 'r') as file:
            data = json.load(file)
    except FileNotFoundError:
        return [np.nan, np.nan, np.nan, np.nan, np.nan]

    try:
        result = data['observations']
        for observation in result:
            hour = datetime.strptime(observation['obsTimeLocal'],
                                     '%Y-%m-%d %H:%M:%S').hour
            if hour == timestamp.hour:
                temp_avg = observation['metric']['tempAvg']
                wind_speed_avg = observation['metric']['windspeedAvg']
                wind_dir_avg = observation['metric']['winddirAvg']
                pressure_max = observation['metric']['pressureMax']

```

```
        humidity_avg = observation['humidityAvg']

        return [temp_avg, wind_speed_avg, wind_dir_avg,
                pressure_max, humidity_avg]
    except KeyError:
        return [np.nan, np.nan, np.nan, np.nan, np.nan]
```


2. Przygotowanie danych do modelowania

Przed przystąpieniem do modelowania przygotowaliśmy dane, w taki sposób, aby mogły być podstawione do modeli. Do zadania tego zostały wykorzystane narzędzia inżynierii danych takie jak:

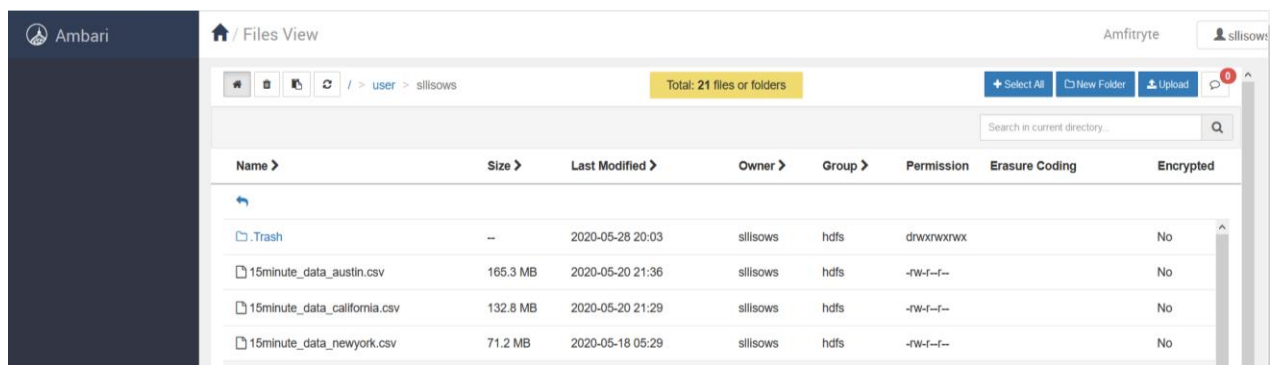
- notatnik Apache Zeppelin – środowisko programistyczne zawierające interpreter języka pyspark,
- Anaconda Spyder – środowisko programistyczne zawierające interpreter języka python,
- Apache Ambari – interfejs web do zarządzania klastrami Hadoop.

Ze względu na duży rozmiar plików z danymi i samych danych, konieczne było wykorzystanie narzędzi używanych najczęściej do przetwarzania, dużych zbiorów danych Big Data. Dzięki użyciu tych narzędzi operacje na danych oraz ich połączenia przebiegały dużo szybciej niż przy użyciu jedynie języka Python. Poniżej zostały przedstawione poszczególne etapy przygotowania danych do ostatecznej analizy.

2.1 Etapy przygotowania danych

2.1.1 Umieszczenie danych poprzez interfejs Apache Ambari w HDFS (Hadoop Distributed File System)

Na klastrze zostały umieszczone trzy pliki csv z danymi zawierającymi obserwacje dotyczące zużycia prądu w budynkach. Każdy plik dotyczy innej lokalizacji.



Rys. 2.1 Widok HDFS interfejsu Apache Ambari wraz z dołączonymi plikami

źródło: opracowanie własne

Po sprawdzeniu struktury okazało się, że wiele zmiennych nie zawiera żadnych danych lub jest ich bardzo niewiele.



Rys. 2.2. Heatmapa wygenerowana za pomocą funkcji `missingno.matrix()` pokazująca obrazowo braki w danych na przykładzie miejscowości Austin

źródło: opracowanie własne

Do dalszej „obróbki” zostały wybrane kolumny z danymi:

- „dataid” – unikalne numery id budynków, z których pochodziły obserwacje,
- „local_15min” - data i czas odczytu liczników prądu w budynkach,
- „grid” – zużycie prądu liczone w odstępach 15 minutowych (zmienna celu).

Pliki zostały połączone w jeden plik „csv” zawierający 2120409 obserwacji.

```
%pyspark
#połączenie danych w jedną ramkę
import functools
def unionAll(dfs):
    return functools.reduce(lambda df1,df2:
df1.union(df2.select(df1.columns)), dfs)
data_all_regions=unionAll([data_austin, data_california,
data_newyork])
```

2.1.2 Dodanie zmiennych ze zbioru metadanych

Na podstawie pliku „metadata.csv”, który zawierał informacje opisujące każdy budynek do ramki danych zostały dołączone nowe zmienne:

- „city” – informacja na temat miejscowości położenia budynku,

- „state – informacja na temat stanu (USA), w którym znajdował się budynek,
- „building_type – informacja na temat rodzaju budynku, zawierała wartości kategoryczne: „Single Family Home (001)”, „Town Home”, „Apartment”,
- - „total_square_footage” – całkowita powierzchnia nieruchomości w stopach.

```
%pyspark

#funkcja tworząca mapę klucz - wartość z dwóch kolumn podanej ramki
danych

def to_dict(df, key, value):
    dict=df.rdd
    key_value=dict.map(lambda x: (x[key], x[value]))
    ready_dict=key_value.collectAsMap()
    mapping = create_map([lit(x) for x in chain(* ready_dict.items())])
    return mapping

#stworzenie map klucz - wartość na podstawie metadanych z pliku
metadata.csv

mapping_city=to_dict(metadata_df, 'dataid', 'city')
mapping_state=to_dict(metadata_df, 'dataid', 'state')
mapping_building=to_dict(metadata_df, 'dataid', 'building_type')
mapping_footage=to_dict(metadata_df, 'dataid', 'total_square_footage')

#dołączenie nowych kolumn do zbioru danych

data_before_weather=data_before_weather.withColumn('city',mapping_city
[data_before_weather['dataid']].alias('city'))\

.withColumn('state',
mapping_state[data_before_weather['dataid']].alias('state'))\

.withColumn('building_type',
mapping_building[data_before_weather['dataid']].alias('building_type')
)\

.withColumn('total_square_footage',
```

2.1.3 Pobranie oraz dołączenie do ramki informacji geolokalizacyjnych i danych pogodowych

Dla każdej lokalizacji zostały pobrane dane geolokalizacyjne czyli długość i szerokość geograficzną miasta, w którym znajdował się budynek. Do tego celu posłużył serwis Google Maps Platform. Za pomocą API tego serwisu zostały pobrane dane geolokalizacyjne dla każdej miejscowości, dzięki, którym, w dalszym kroku mogła zostać określona lokalizacja stacji pogodowej do pobrania danych meteo.

#funkcja pozwalająca określić położenie geograficzne (długość i szerokość geograficzną) na podstawie nazwy miejscowości i stanu.

```
def geolocation_per_city(city, state):
    location = city + ' ' + state
    api_key = GOOGLE_API
    payload = {'address': location, 'key': api_key}
    headers = {'Accept-Encoding': 'gzip'}
    url = 'https://maps.googleapis.com/maps/api/geocode/json'
    r = requests.get(url=url, params=payload, headers=headers)
    result = r.json()
    latitude =
round(result['results'][0]['geometry']['location']['lat'], 2)
    longitude =
round(result['results'][0]['geometry']['location']['lng'], 2)
    return [latitude, longitude]
```

Po uzyskaniu danych geolokalizacyjnych do ramki danych zostały dołączone odpowiednie kolumny.

dataid	local_15min	grid	city	lat	lng	state	building_type	total_square_footage
661 2018-11-21 15:15:00		0.124	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 15:30:00		0.251	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 15:45:00		0.419	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 16:00:00		0.833	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 16:15:00		1.105	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 16:30:00	0.8690000000000001		Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 16:45:00		1.324	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 17:00:00		0.993	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 17:15:00		0.906	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0
661 2018-11-21 17:30:00		1.203	Austin	30.27	-97.74	Texas	Single-Family Hom...	1842.0

Rys. 2.3. Widok ramki danych po dodaniu danych geolokalizacyjnych („lat” – dł. geograficzna, „lng” – szerokość geograficzna)

źródło: opracowanie własne, dane geolokalizacyjne pobrane ze strony <https://maps.googleapis.com/maps/api/geocode/json>

Docelowa ramka danych miała również posiadać zmienne zawierające dane pogodowe. Informacje pogodowe miały pokazywać podstawowe dane meteorologiczne o godzinie, o której dokonywany był pomiar poboru energii elektrycznej. Dane dotyczące pomiarów zostały pobrane z serwisu <https://www.wunderground.com/>.

```

root
|-- observations: array (nullable = true)
|   |-- element: struct (containsNull = true)
|   |   |-- epoch: long (nullable = true)
|   |   |-- humidityAvg: long (nullable = true)
|   |   |-- humidityHigh: long (nullable = true)
|   |   |-- humidityLow: long (nullable = true)
|   |   |-- lat: double (nullable = true)
|   |   |-- lon: double (nullable = true)
|   |   |-- metric: struct (nullable = true)
|   |   |   |-- dewptAvg: long (nullable = true)
|   |   |   |-- dewptHigh: long (nullable = true)
|   |   |   |-- dewptLow: long (nullable = true)
|   |   |   |-- heatindexAvg: string (nullable = true)
|   |   |   |-- heatindexHigh: string (nullable = true)
|   |   |   |-- heatindexLow: string (nullable = true)
|   |   |   |-- precipRate: string (nullable = true)
|   |   |   |-- precipTotal: double (nullable = true)
|   |   |   |-- pressureMax: double (nullable = true)
|   |   |   |-- pressureMin: double (nullable = true)
|   |   |   |-- pressureTrend: double (nullable = true)
|   |   |   |-- tempAvg: long (nullable = true)
|   |   |   |-- tempHigh: long (nullable = true)
|   |   |   |-- tempLow: long (nullable = true)
|   |   |   |-- windchillAvg: string (nullable = true)
|   |   |   |-- windchillHigh: string (nullable = true)
|   |   |   |-- windchillLow: string (nullable = true)
|   |   |   |-- windgustAvg: string (nullable = true)
|   |   |   |-- tempAvg: long (nullable = true)
|   |   |   |-- tempHigh: long (nullable = true)
|   |   |   |-- tempLow: long (nullable = true)
|   |   |   |-- windchillAvg: string (nullable = true)
|   |   |   |-- windchillHigh: string (nullable = true)
|   |   |   |-- windchillLow: string (nullable = true)
|   |   |   |-- windgustAvg: string (nullable = true)
|   |   |   |-- windgustHigh: string (nullable = true)
|   |   |   |-- windgustLow: string (nullable = true)
|   |   |   |-- windspeedAvg: long (nullable = true)
|   |   |   |-- windspeedHigh: long (nullable = true)
|   |   |   |-- windspeedLow: long (nullable = true)
|   |   |   |-- obsTimeLocal: string (nullable = true)
|   |   |   |-- obsTimeUtc: string (nullable = true)
|   |   |   |-- qcStatus: long (nullable = true)
|   |   |   |-- solarRadiationHigh: string (nullable = true)
|   |   |   |-- stationID: string (nullable = true)
|   |   |   |-- tz: string (nullable = true)
|   |   |   |-- uvHigh: string (nullable = true)
|   |   |   |-- winddirAvg: long (nullable = true)

```

Rys. 2.4. Struktura pliku JSON z danymi pogodowymi dla danej lokalizacji

źródło: opracowanie własne, plik JSON pobrany ze strony <https://api.weather.com/v2/pws/history/hourly>

Po założeniu konta w serwisie i wygenerowaniu klucza API, była możliwość pobrania plików JSON z historycznymi danymi pogodowymi dla stacji pogodowych znajdujących się w pobliżu miejscowości. Po ustaleniu nazwy stacji pogodowej dla konkretnej daty i lokalizacji, zostały pobrane pliki, które zawierały obserwacje pogodowe w odstępach jednej godziny.

Aby dostosować pomiary poboru energii elektrycznej do godzinnych odczytów danych meteo, dane w kolumnie „grid” zostały przekształcone w taki w taki sposób, aby, pokazywały sumaryczne zużycie energii elektrycznej w ostatniej godzinie. Do osiągnięcia tego celu została zastosowana funkcja „window” z pakietu `pyspark.sql.functions`.

```
#użycie funkcji window do agregacji pomiarów 15 minutowych w pomiary
co godzinę
df_new = data_before_weather.groupBy("dataid", window("local_15min",
"1 hour")).sum("grid")
```

Dzięki temu zabiegowi znacząco zmniejszyła się również ilość danych. Pomiar z czterech okresów czasowych zostały zagregowane w jeden.

Kolejnym krokiem było wczytanie danych pogodowych z pobranych plików JSON. Z wielu informacji pogodowych zostały wybrane:

- „tempAvg” – średnia temperatura powietrza w stopniach Celsjusza,
- „windspeedAvg” – średnia prędkość wiatru w m/s,

- „windirAvg”- uśredniony pomiar kierunku wiatru,
- „pressureMax” – maksymalna wartość ciśnienia atmosferycznego w hPa,
- „humidityAvg” – średnia wilgotność powietrza z danego dnia w %.

	dataid	local_1hour	state	building_type	total_square_footage	lat	lng	temp_avg	wind_speed_avg	wind_dir_avg	pressure_max	humidity_avg	grid
0	203	2015-01-01 02:00:00	California	Single-Family Home 001 (Master)	1555.0	32.72	-117.16	4.0	5.0	248.0	1017.95	75.0	1.234
1	203	2015-01-01 03:00:00	California	Single-Family Home 001 (Master)	1555.0	32.72	-117.16	4.0	4.0	235.0	1017.95	74.0	1.411
2	203	2015-01-01 04:00:00	California	Single-Family Home 001 (Master)	1555.0	32.72	-117.16	4.0	4.0	97.0	1015.58	78.0	1.509
3	203	2015-01-01 05:00:00	California	Single-Family Home 001 (Master)	1555.0	32.72	-117.16	3.0	4.0	113.0	1016.59	79.0	0.834
4	203	2015-01-01 06:00:00	California	Single-Family Home 001 (Master)	1555.0	32.72	-117.16	4.0	5.0	245.0	1018.96	74.0	0.794

Rys. 2.5. Widok zbioru danych z umieszczonymi danymi meteo

źródło: opracowanie własne, dane pogodowe pobrane ze strony <https://api.weather.com/v2/pws/history/hourly>, dane geolokalizacyjne pobrane ze strony <https://maps.googleapis.com/maps/api/geocode/json>

2.1.4 Pozostałe przekształcenia danych

W dalszej kolejności Kolumny z danymi zawierającymi stan („state”), oraz rodzaj budynku („building_type”), zostały przedstawione w postaci binarnej. Kolumna „state” zawierała dwie unikalne wartości – „Texas” i „California” (po usunięciu z danych rekordów, które nie zawierały informacji na temat powierzchni nieruchomości – „total_square_footage” dane ze stanu Nowy Jork zostały usunięte) , natomiast kolumna „building_type”, trzy unikalne wartości - „Single Family Home (001)”, „Town Home”, „Apartment”. Aby zakodować binarnie tą kolumnę została użyta funkcja „OneHotEncoder” z biblioteki sklearn.preprocessing języka Python.

```
from sklearn.preprocessing import OneHotEncoder

#stworzenie tablicy z wartościami 0 i 1 dla zmiennej building_type
enc = OneHotEncoder(handle_unknown='ignore')

enc_df =
pd.DataFrame(enc.fit_transform(data_weather[['building_type']],
).toarray(), columns=['Apartment', 'Single_Family_Home', 'Town_Home'])

data_weather=data_weather.join(enc_df).copy()
```

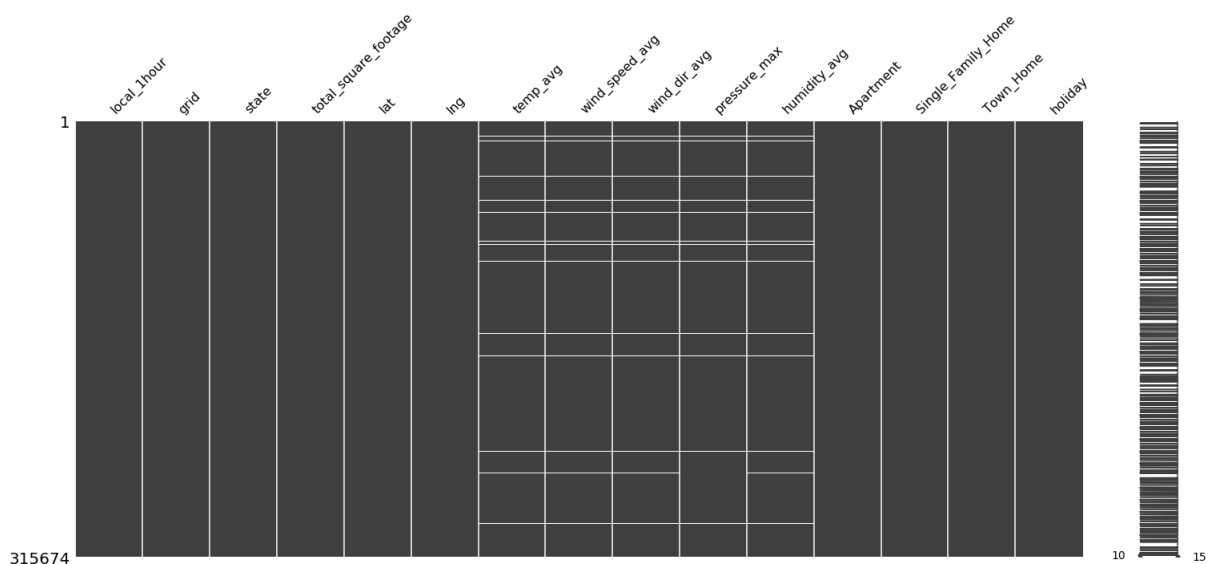
Na koniec do ramki danych została dodana informacja na temat tego czy dany dzień był dniem wolnym od pracy w Stanach Zjednoczonych. Została do tego użyta biblioteka „holidays” języka Python, która zawiera informacje na temat dni wolnych od pracy w Stanach Zjednoczonych. Powstała dodatkowa zmienna „holiday”.

	local_1hour	state	total_square_footage	lat	lng	temp_avg	wind_speed_avg	wind_dir_avg	pressure_max	humidity_avg	Apartment	Single_Family_Home	Town_Home	holiday	grid
0	2015-01-01 02:00:00	0	1555.0	32.72	-117.16	4.0	5.0	248.0	1017.95	75.0	0.0	1.0	0.0	1	1.234
1	2015-01-01 03:00:00	0	1555.0	32.72	-117.16	4.0	4.0	235.0	1017.95	74.0	0.0	1.0	0.0	1	1.411
2	2015-01-01 04:00:00	0	1555.0	32.72	-117.16	4.0	4.0	97.0	1015.58	78.0	0.0	1.0	0.0	1	1.509
3	2015-01-01 05:00:00	0	1555.0	32.72	-117.16	3.0	4.0	113.0	1016.59	79.0	0.0	1.0	0.0	1	0.834
4	2015-01-01 06:00:00	0	1555.0	32.72	-117.16	4.0	5.0	245.0	1018.96	74.0	0.0	1.0	0.0	1	0.794

Rys. 2.6. Widok ramki danych przed przystąpieniem do ostatecznej analizy

źródło: opracowanie własne, dane pogodowe pobrane ze strony <https://api.weather.com/v2/pws/history/hourly>, dane geolokalizacyjne pobrane ze strony <https://maps.googleapis.com/maps/api/geocode/json>

Na końcowym etapie uzyskaliśmy ramkę danych z 14 zmiennymi opisującymi oraz zmienną celu „grid”. Na rysunku 2.7 możemy porównać strukturę danych jeżeli chodzi o braki z początkową ramką danych.



Rys. 2.7. Heatmapa wygenerowana za pomocą funkcji missingno.matrix() pokazująca obrazowo braki w danych

źródło: opracowanie własne

3. EDA - Eksploracyjna Analiza Danych

3.1 Importowanie potrzebnych bibliotek oraz pakietów

```
library(tidyverse)
library(lubridate)
library(corrplot)
library(kableExtra)
library(leaflet)
library(scales)
library(tmap)
library(maps)
```

3.2 Importowanie danych oraz łączenie w jeden zbiór

```
dane_1a <- read.csv("data/pipeline_661_austin.csv")
dane_1b <- read.csv("data/pipeline_1642_austin.csv")
dane_1c <- read.csv("data/pipeline_2335_austin.csv")
dane_2a <- read.csv("data/pipeline_2358_ny.csv")
dane_2b <- read.csv("data/pipeline_4550_ny.csv")
dane_2c <- read.csv("data/pipeline_558_ny.csv")
dane_3a <- read.csv("data/pipeline_3687_california.csv")
dane_3b <- read.csv("data/pipeline_6377_california.csv")
dane_3c <- read.csv("data/pipeline_7062_california.csv")
```

Importowanie danych dla 3 różnych lokalizacji w każdym z 3 stanów: Texas, California, Nowy Jork. Wybrany zbiór jest reprezentatywny dla całego datasetu. Głównym pakietem używanym przy tworzeniu eksploracyjnej analizy danych jest *tidyverse*. Jest to w zasadzie zbiór pakietów najbardziej potrzebnych przy analizie danych. Dwa główne komponenty tego pakietu to *dplyr* oraz *ggplot2*. W niniejszej analizie są one używane na każdym etapie tworzenia wykresów, najpierw tworzymy ograniczony zbiór danych **temp** za pomocą tzw. pipe'a `%>%` a następnie wykres za pomocą funkcji `ggplot`.

```
temp <- dane_all %>% select(grid, state)

ggplot(temp, aes(grid)) +
  geom_histogram(bins = 50, aes(y = ..density..), fill = "blue") +
  geom_density(alpha = 0.002, fill = "black") +
  ggtitle("Rozkład zużycia prądu") +
  geom_vline(xintercept = round(mean(temp$grid), 2), size = 1, linetype = 2) +
  facet_wrap(~ state, ncol = 2)
```

3.3 Spojrzenie na strukturę danych:

```
dane_all %>% str()
```



```
## 'data.frame':    261735 obs. of  15 variables:
## $ X              : int  0 1 2 3 4 5 6 7 8 9 ...
## $ dataid         : int  661 661 661 661 661 661 661 661 661 661 661 ...
## $ local_15min    : chr  "2018-11-21 15:15:00" "2018-11-21 15:30:00" "2018-11-21 15:45:00" "2018-11-21 16:00:00" ...
## $ grid           : num  0.124 0.251 0.419 0.833 1.105 ...
## $ city           : chr  "Austin" "Austin" "Austin" "Austin" .
## ..
## $ state          : chr  "Texas" "Texas" "Texas" "Texas" ...
## $ station_id     : chr  "KTXAUSTI1619" "KTXAUSTI1619" "KTXAUSTI1619" "KTXAUSTI1619" ...
## $ latitude       : num  30.3 30.3 30.3 30.3 30.3 ...
## $ longitude      : num  -97.7 -97.7 -97.7 -97.7 -97.7 ...
## $ temp_avg       : num  13 13 13 12 12 12 12 12 12 12 ...
## $ wind_speed_avg : num  0 0 0 0 0 0 0 0 0 0 ...
## $ wind_dir_avg   : num  346 346 346 292 292 292 292 277 277 277 ...
## $ pressure_max   : num  1021 1021 1021 1021 1021 ...
## $ humidity_avg   : num  64 64 64 67 67 67 67 72 72 72 ...
## $ holiday        : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1 1 1 ...
```

3.3.1 Podsumowanie:

```
dane_all %>% summary()
```

```
##           X              dataid      local_15min           grid
## Min.      :    0   Min.      : 558   Length:261735   Min.      :-
## 4.7940
## 1st Qu.:  9349   1st Qu.:1642   Class :character   1st Qu.:
## 0.2430
## Median : 19473   Median :2358   Mode  :character   Median :
## 0.4340
## Mean    : 29484   Mean    :3408                      Mean    :
## 0.5851
## 3rd Qu.: 38713   3rd Qu.:6377                      3rd Qu.:
## 0.8060
## Max.    :104147   Max.    :7062                      Max.    :1
## 0.4640
##
##           city              state           station_id
## Length:261735   Length:261735   Length:261735   Min
## :30.27
```

```

## Class :character   Class :character   Class :character   1st
Qu.:30.27
## Mode :character   Mode :character   Mode :character   Med
ian :32.72
##
n :33.71
##
Qu.:32.72
##
. :42.44
##
## longitude          temp_avg          wind_speed_avg      wind_dir
_avg
## Min. : -117.16    Min. : -33.0    Min. : 0.000    Min. :
0
## 1st Qu.: -117.16    1st Qu.: 15.0    1st Qu.: 1.000    1st Qu.:
96
## Median : -97.74    Median : 19.0    Median : 2.000    Median :2
19
## Mean : -101.23    Mean : 19.6    Mean : 3.415    Mean :1
96
## 3rd Qu.: -97.74    3rd Qu.: 24.0    3rd Qu.: 5.000    3rd Qu.:2
86
## Max. : -76.39    Max. : 44.0    Max. :25.000    Max. :3
60
##
NA's :7104    NA's :7104    NA's :7
104
## pressure_max      humidity_avg      holiday
## Min. : 937.7    Min. : 5.00    False:177944
## 1st Qu.:1008.1    1st Qu.:56.00    True : 83791
## Median :1013.5    Median :74.00
## Mean :1010.1    Mean :70.01
## 3rd Qu.:1016.6    3rd Qu.:87.00
## Max. :1044.7    Max. :99.00
## NA's :3856    NA's :7104

```

3.3.2 Wnioski:

Zbiór posiada 34 700 obserwacji dla każdego domu oraz 15 różnych zmiennych:

- **X** - liczba porządkowa
- **dataid** - id dla obserwacji, oznacza powiązanie dom - mieszkańcy
- **local_15min** - zmienna daty, rok - miesiąc - dzień godzina:minuty

- **grid** - zużycie prądu mierzone co 15 minut
- **city** - miasto
- **state** - stan
- **station_id** - id stacji pogodowej, z której pobrano dane pogodowe w danym momencie czasu
- **latitude** - szerokość geograficzna
- **longitude** - długość geograficzna
- **temp_avg** - średnia wartość temperatury w stopniach C
- **wind_speed_avg** - średnia siła wiatru
- **wind_dir_avg** - średni kierunek wiatru
- **pressure_max** - maksymalna wartość ciśnienia
- **humidity_avg** - wilgotność
- **holiday** - zmienna jakościowa oznaczająca czy dany dzień jest świętem

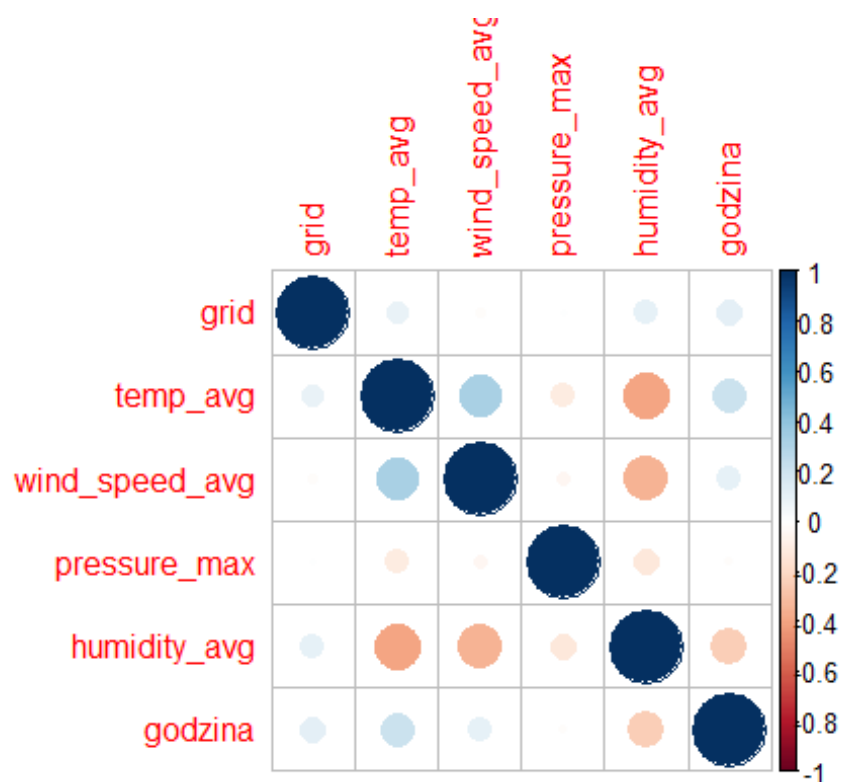
3.3.3 Spojrzenie na 6 pierwszych wierszy

```
dane_all %>% head()
```

```
##   X dataid          local_15min  grid   city state  station_i
d latitude
## 1 0      661 2018-11-21 15:15:00 0.124 Austin Texas KTXAUSTI161
9    30.27
## 2 1      661 2018-11-21 15:30:00 0.251 Austin Texas KTXAUSTI161
9    30.27
## 3 2      661 2018-11-21 15:45:00 0.419 Austin Texas KTXAUSTI161
9    30.27
## 4 3      661 2018-11-21 16:00:00 0.833 Austin Texas KTXAUSTI161
9    30.27
## 5 4      661 2018-11-21 16:15:00 1.105 Austin Texas KTXAUSTI161
9    30.27
## 6 5      661 2018-11-21 16:30:00 0.869 Austin Texas KTXAUSTI161
9    30.27
##   longitude temp_avg wind_speed_avg wind_dir_avg pressure_max
humidity_avg
## 1    -97.74      13           0           346      1021.33
64
## 2    -97.74      13           0           346      1021.33
64
## 3    -97.74      13           0           346      1021.33
64
## 4    -97.74      12           0           292      1021.33
67
## 5    -97.74      12           0           292      1021.33
67
```

```
## 6      -97.74      12      0      292      1021.33
67
##      holiday
## 1      False
## 2      False
## 3      False
## 4      False
## 5      False
## 6      False
```

3.4 Analiza korelacji zmiennych liczbowych



3.4.1 Wnioski:

- negatywna korelacja między temperaturą a ciśnieniem (< -0.3)
- negatywna korelacja między siłą wiatru a wilgotnością (< -0.3)
- pozytywna korelacja między prędkością wiatru a temperaturą (0.3)
- pozytywna korelacja między zmienną celu grid a wilgotnością (< 0.2)
- pozytywna korelacja między zmienną celu grid a godziną (< 0.2)

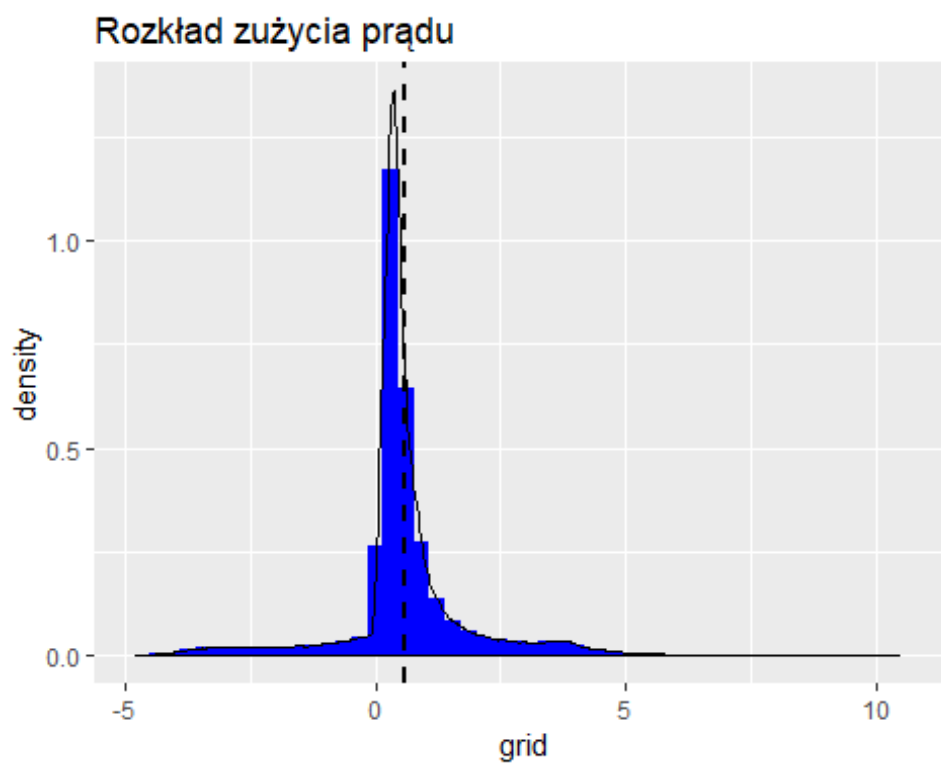
3.5 Analiza zmiennej 'grid' - zużycie prądu

Przy prezentacji rozkładów użyta została funkcja gęstości (density).

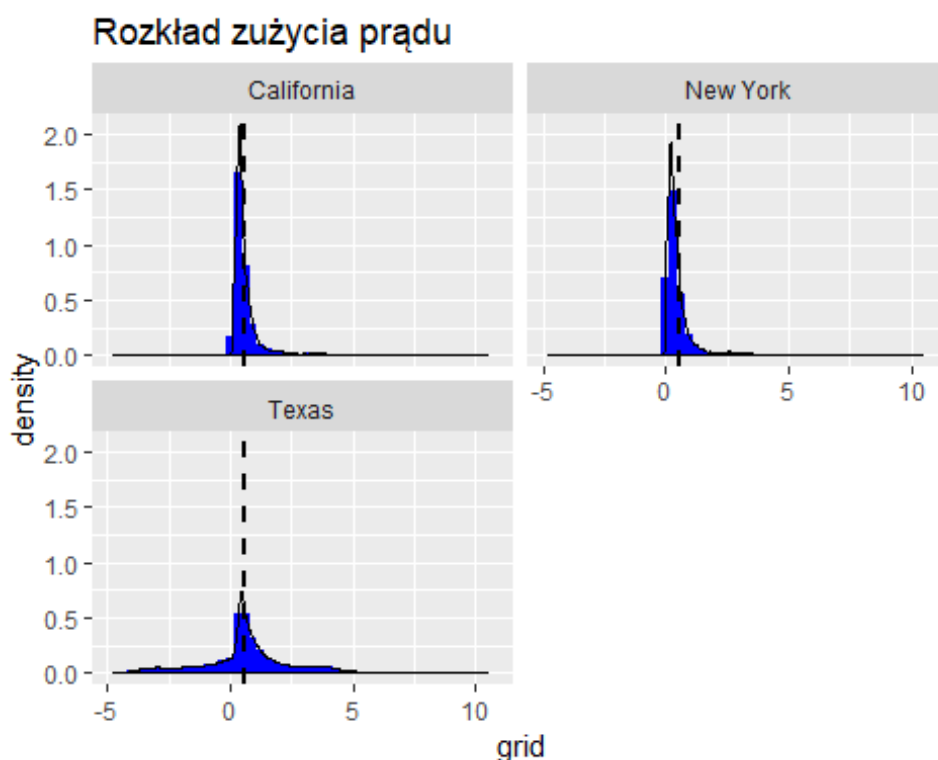
Podział wartości zużycia prądu na kwantyle:

0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-4.794	0.028	0.185	0.286	0.361	0.434	0.532	0.688	0.971	1.835	10.464

3.5.1 Wykres rozkładu zmiennej zużycie prądu



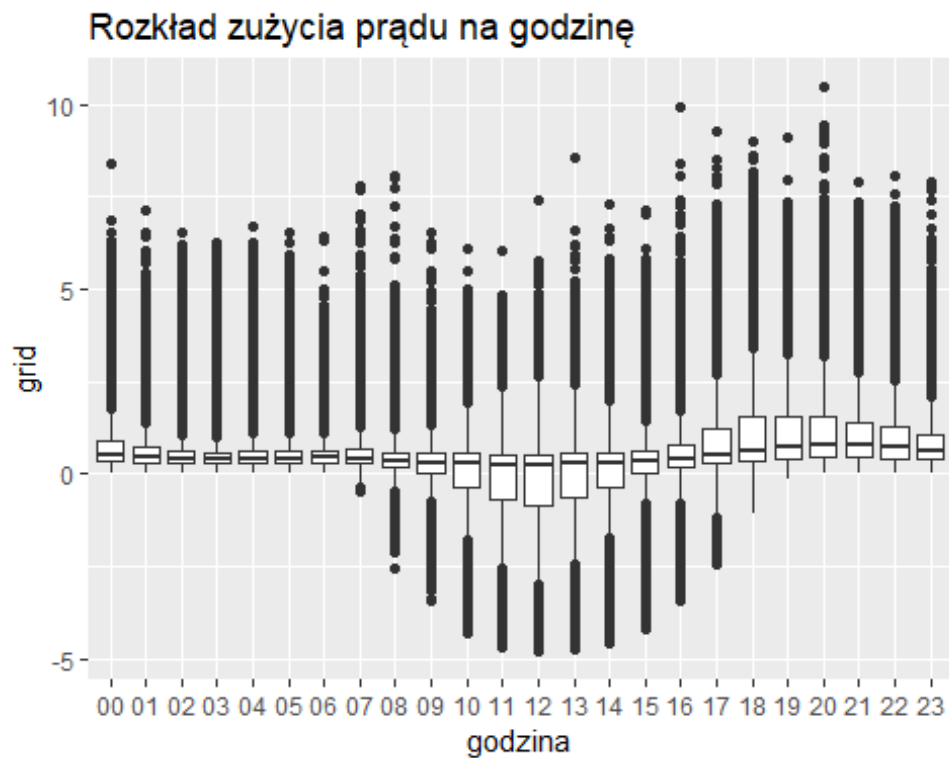
3.5.2 Wykres rozkładu zmiennej zużycie prądu z podziałem na stany



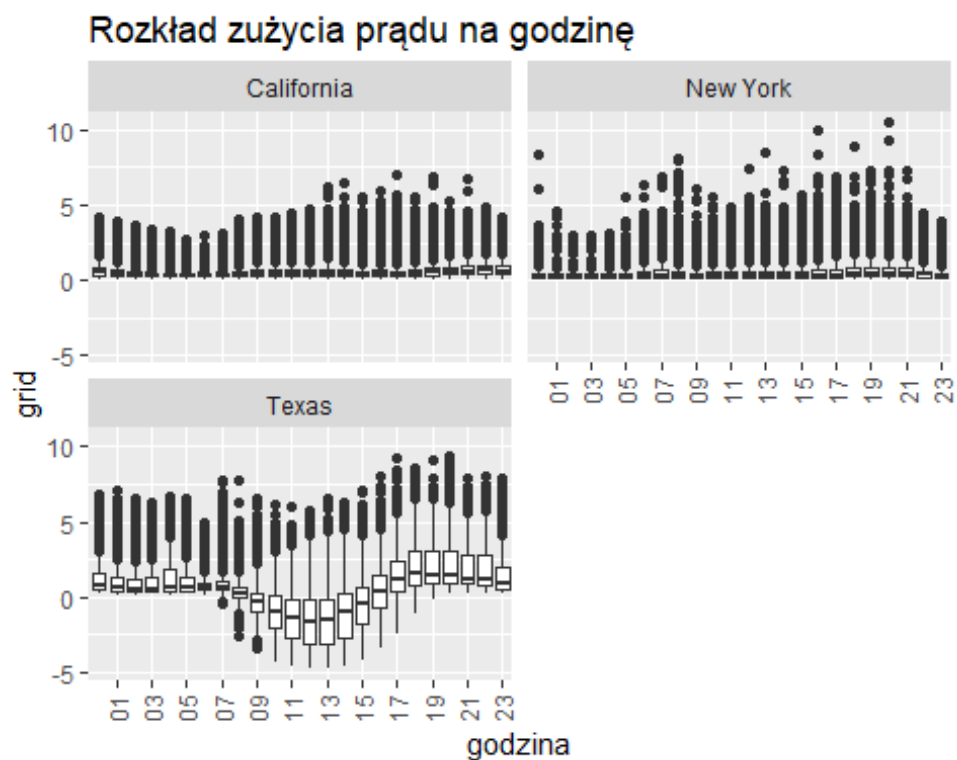
3.5.3 Wnioski:

- Analiza rozkładu zmiennej wskazuje na na duże zagęszczenie wartości wokół średniej. Zdecydowana większość wartości pozostaje w przedziale 0 - 1.
- Dodanie podziału na stan wyjaśnia zagęszczenie w przedziale 0-1, w stanach Kalifornia oraz Nowy Jork zużycie prądu przyjmuje tylko wartości dodatnie - wybrane id-domy nie posiadają własnych źródeł prądu. W wybranych domach w Teksasie panele fotowoltaiczne produkują energię elektryczną.

3.5.4 Wykres pudełkowy prezentujący zużycie prądu w ciągu doby



3.5.5 Wykres pudełkowy prezentujący zużycie prądu w ciągu doby w podziale na stany



3.5.6 Wnioski:

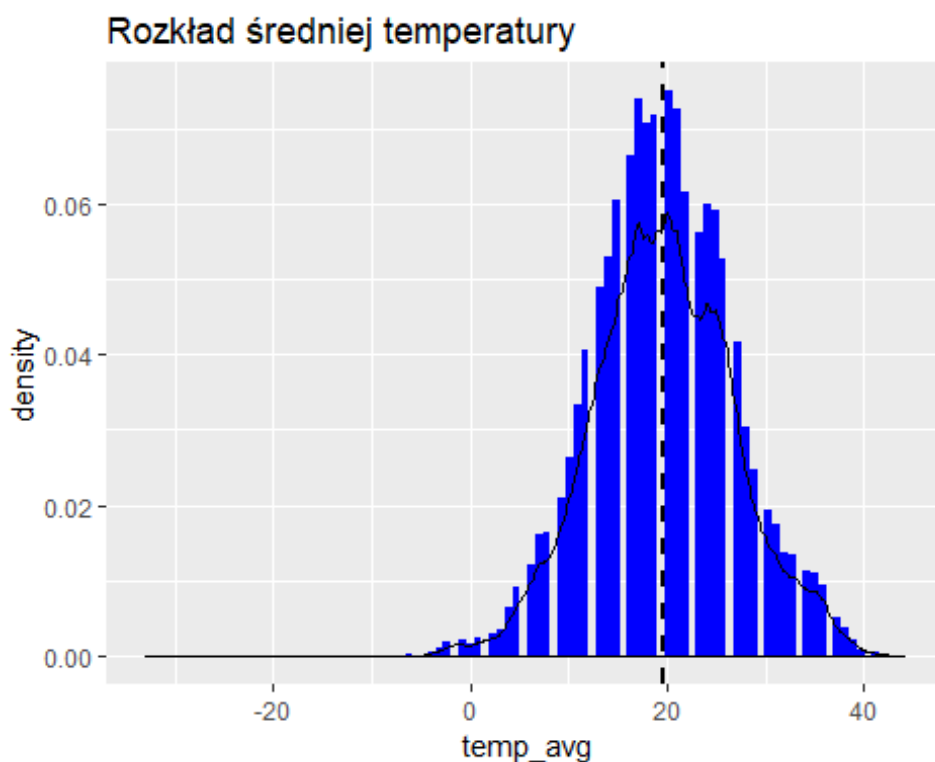
Analiza rozkładów wskazuje na duże podobieństwo dla Kaliforni oraz Nowego Jorku, z niewiele większym rozmyciem danych dla drugiego stanu. W obu przypadkach jednak widać wyższe zużycie w ciągu dnia ze szczytem późnym popołudniem/wieczorem. W stanie Teksas ze względu na występowanie wartości ujemnych rozkład diametralnie się zmienia. Zauważyć można jednak silny wzrost w godzinach popołudniowych, gdy rośnie zużycie, a jednocześnie przez spadek promieniowania słonecznego produkcja energii maleje.

3.6 Analiza zmiennej 'temp_avg' - średniej temperatury

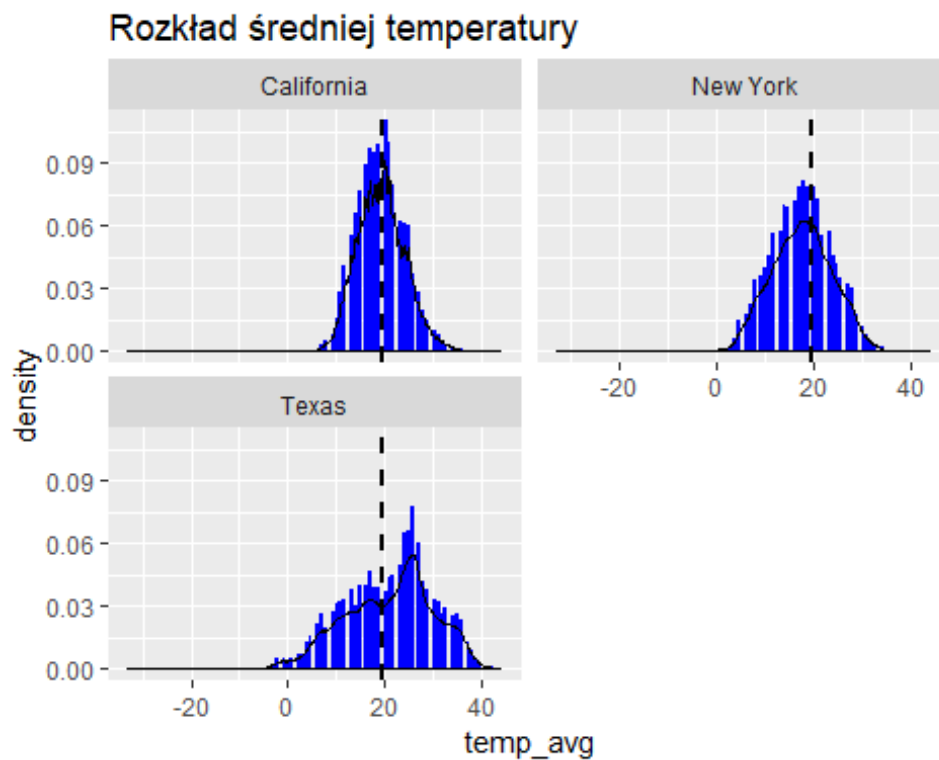
Podział wartości średniej temperatury na kwantyle:

##	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
##	-33	11	14	16	18	19	21	23	26	29	44

3.6.1 Wykres rozkładu średniej temperatury



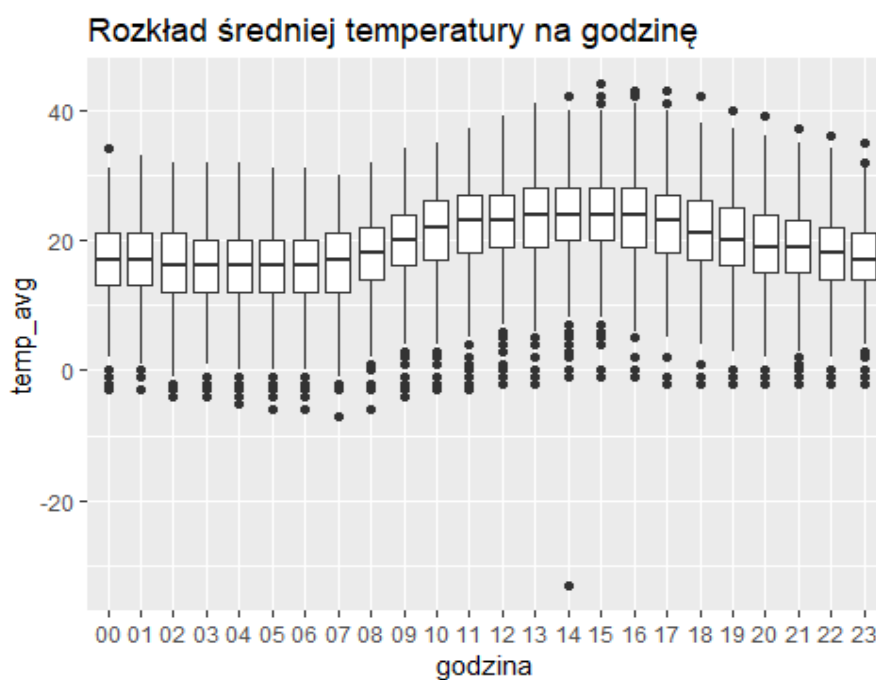
3.6.2 Wykres rozkładu średniej temperatury z podziałem na stany



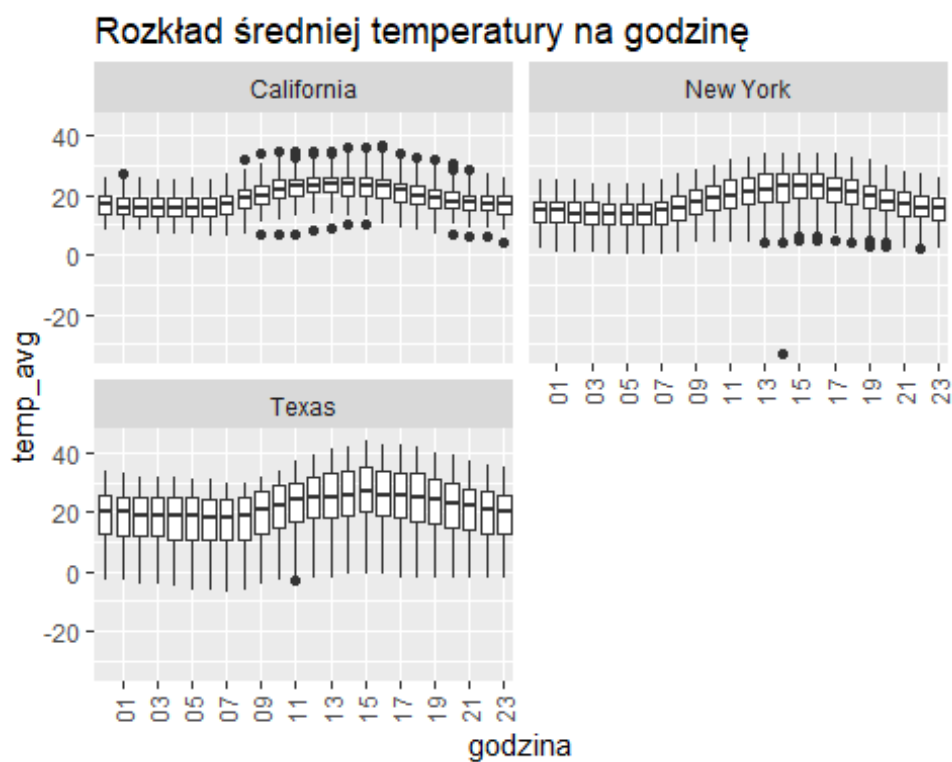
3.6.3 Wnioski:

- Analiza rozkładu zmiennej wskazuje na rozkład normalny. W podziale na stany widać różnice, jednak wartości rozkładają się po obu stronach średniej.
- Dodanie podziału na stany wyjaśnia dodatkowo najmniejsze wahania temperatury w Kalifornii oraz największe w Teksasie.

3.6.4 Wykres pudełkowy prezentujący średnie temperatury w ciągu doby



3.6.5 Wykres pudełkowy prezentujący średnią temperaturę w ciągu doby w podziale na stany



3.6.6 Wnioski:

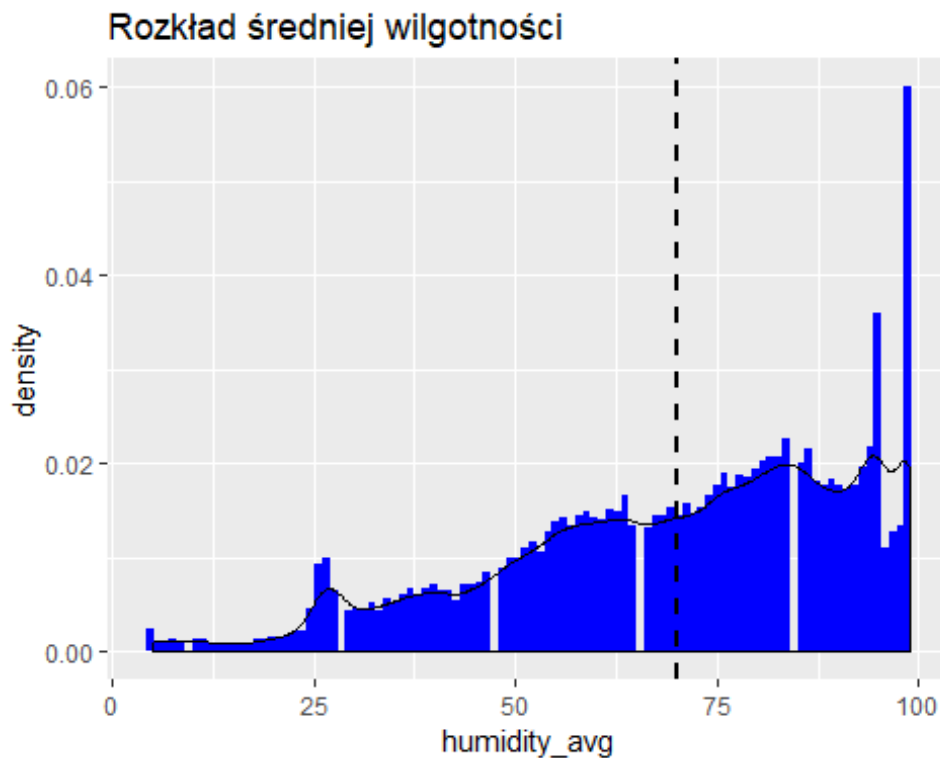
Rozkład średniej temperatury wskazuje niewielkie zmiany dobowe. Podział na stany nie wnosi wiele do obserwacji poza różnicą w amplitudach temperatur, najniższą w Nowym Jorku i najwyższą w Teksasie. Ten ostatni stan charakteryzuje się również na najwyższą zmiennością dobową oraz najwyższymi temperaturami maksymalnymi.

3.7 Analiza zmiennej 'humidity_avg' - średniej wilgotności

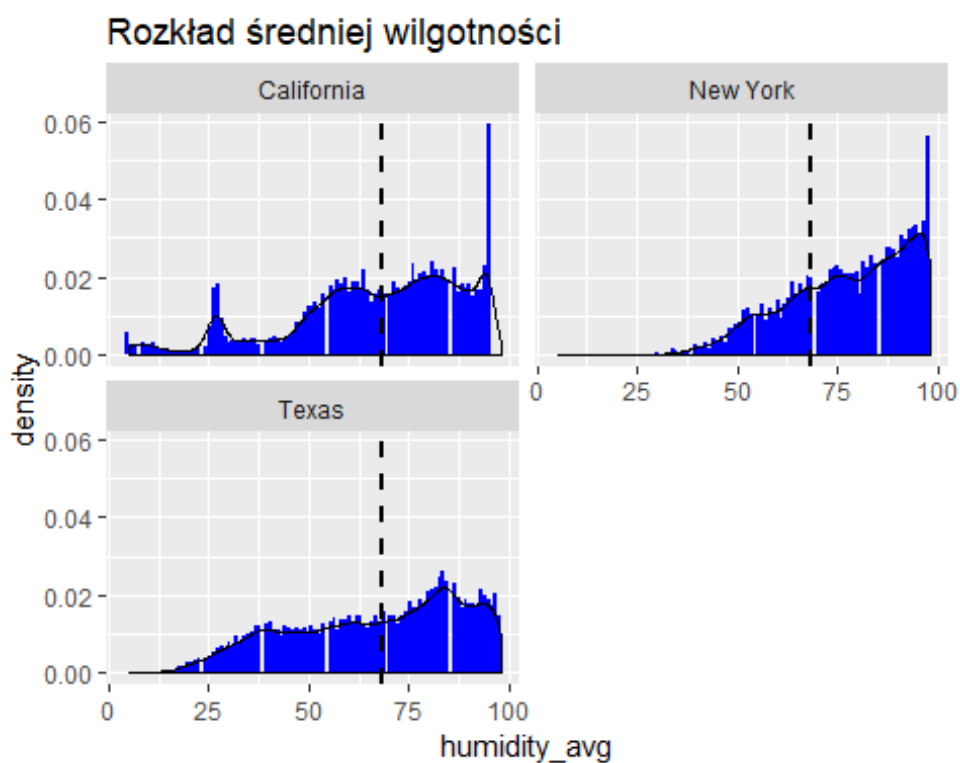
Podział wartości średniej wilgotności na kwantyle:

##	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
##	5	38	52	60	67	74	80	85	90	95	99

3.7.1 Wykres rozkładu średniej wilgotności



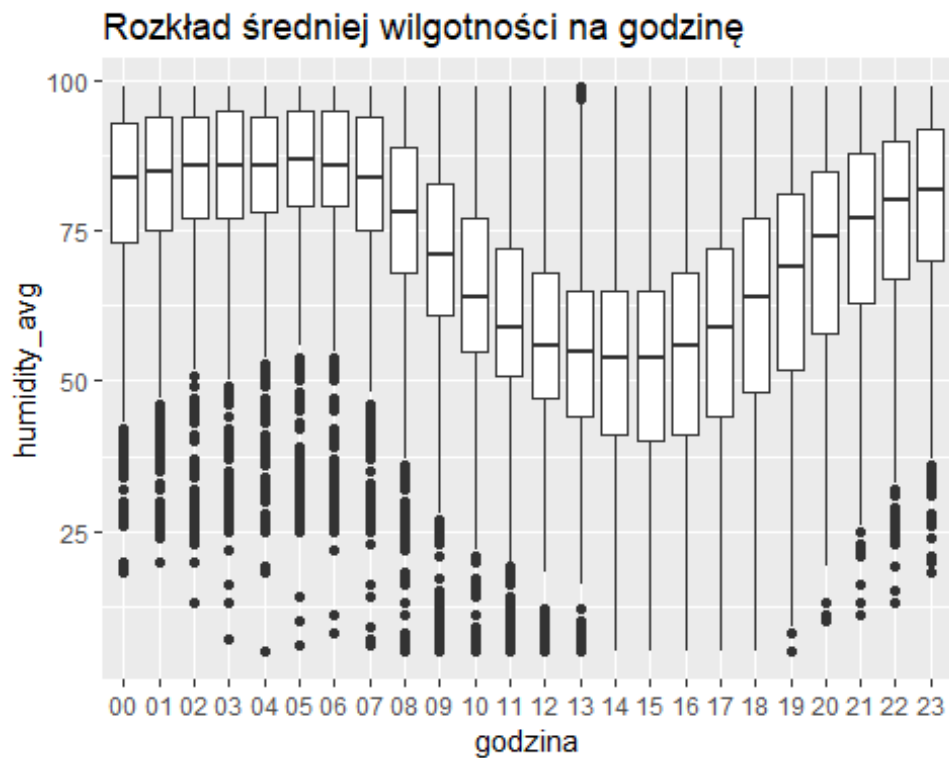
3.7.2 Wykres rozkładu średniej wilgotności z podziałem na stany



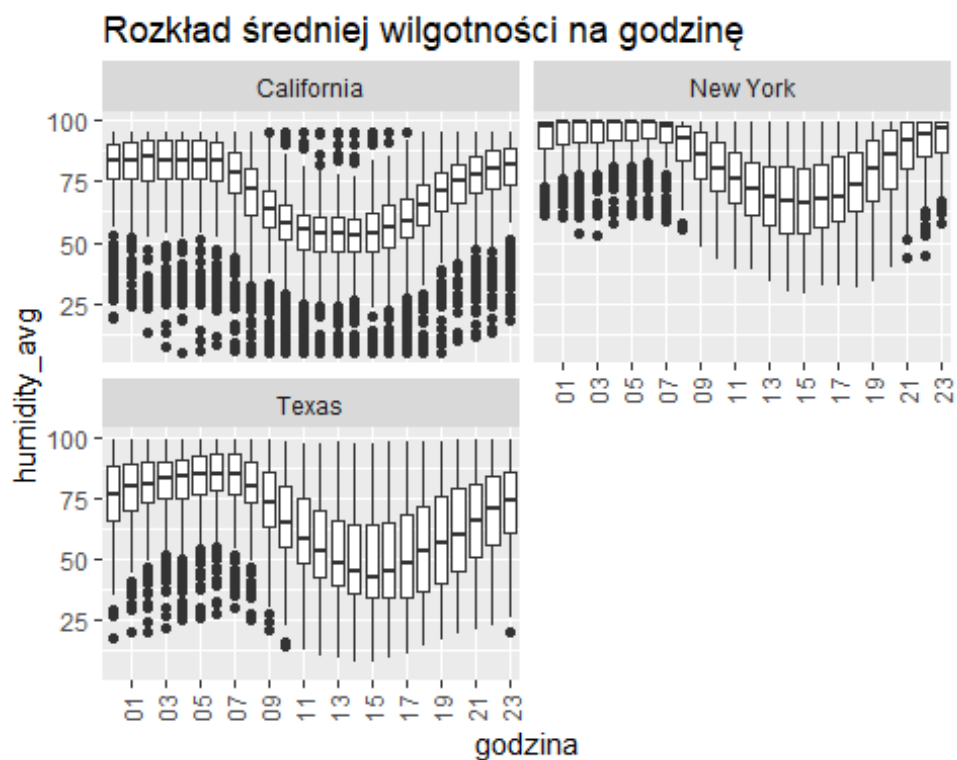
3.7.3 Wnioski:

- Analiza rozkładu zmiennej wskazuje na większą gęstość wartości wilgotności w górze przedziału.
- Dodanie podziału na stany wskazuje najwyższą wilgotność dla Nowego Jorku.

3.7.4 Wykres pudełkowy prezentujący średnią wilgotność w ciągu doby



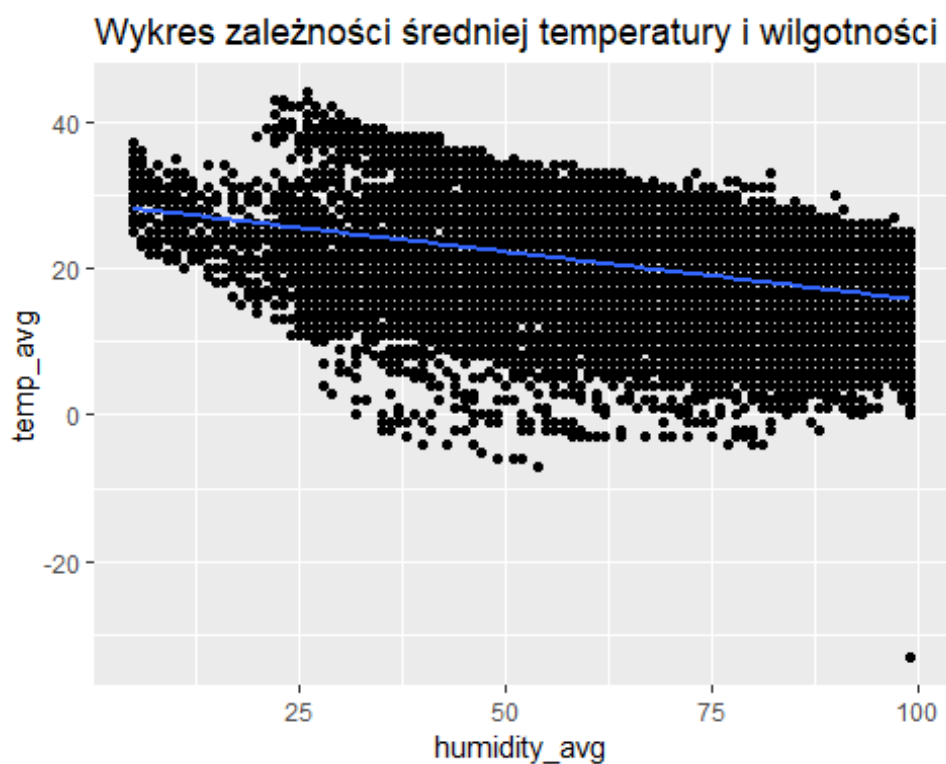
3.7.4 Wykres pudełkowy prezentujący średnią wilgotność w ciągu doby w podziale na stany



3.7.5 Wnioski:

- Rozkład średniej wilgotności wskazuje na silną zależność dobową.
- W przypadku indywidualnych stanów, Kalifornia wykazuje znacznie większe rozmycie wartości skrajnych, przy zachowaniu tendencji dobowej.

3.8 Wskazanie zależności między średnią temperaturą a wilgotnością



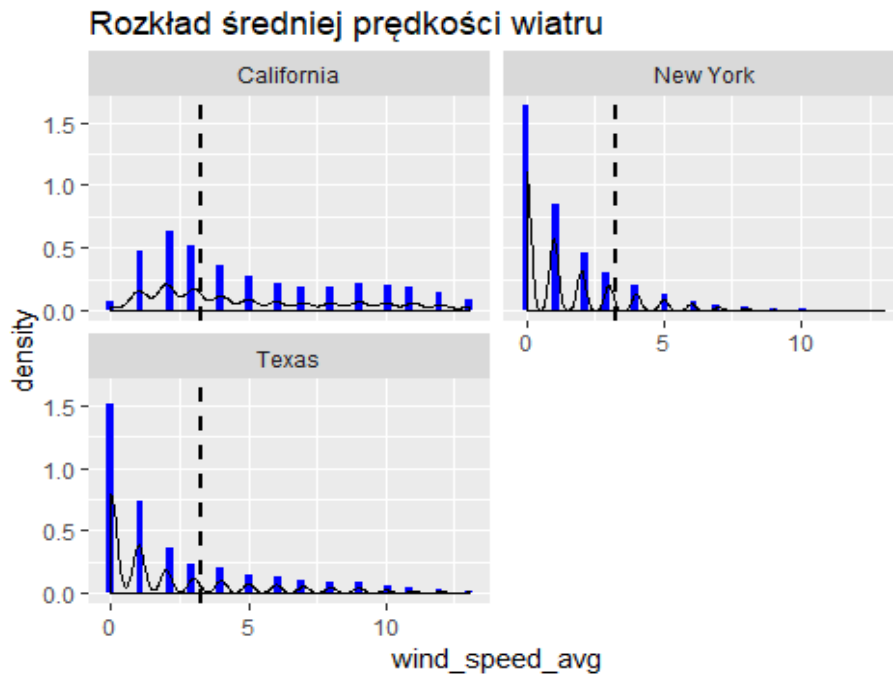
Zauważyć można pozytywną korelację. Jednak rozmycie na wykresie sugeruje iż korelacja nie jest silna.

3.9 Analiza zmiennej 'wind_speed_avg' - średniej prędkości wiatru

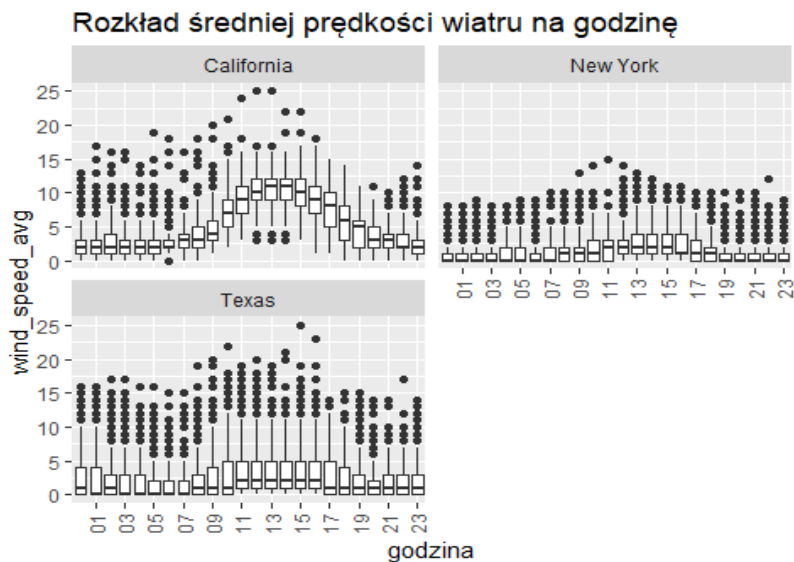
Podział wartości średniej wilgotności na kwantyle:

##	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
##	0	0	0	1	1	2	3	4	6	9	25

3.9.1 Wykres rozkładu średniej prędkości wiatru z podziałem na stany



3.9.3 Wykres pudełkowy prezentujący średnią prędkość wiatru w ciągu doby w podziale na stany

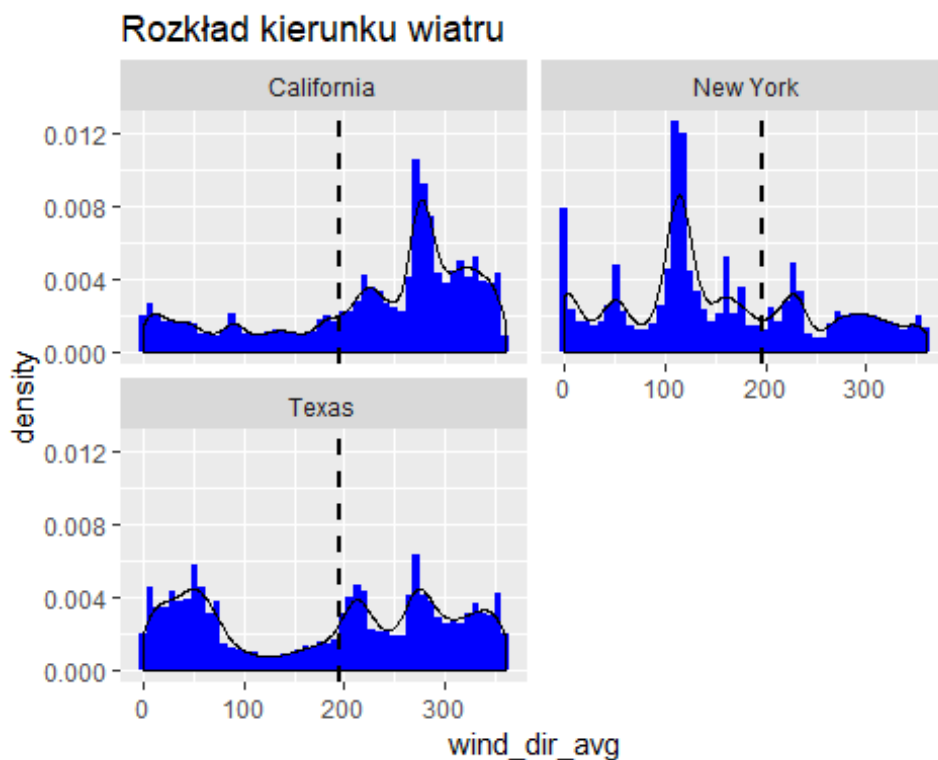


3.9.4 Wnioski:

- Rozkład charakteryzuje się dużą zmiennością godzinową jedynie dla Kalifornii. Średnia siła wiatru znacząco rośnie w godzinach około południowych.
- Analiza rozkładu zmiennej wskazuje na podobieństwo zjawiska w Nowym Jorku oraz Teksasie - większość obserwacji poniżej średniej, często występująca wartość 0. Może to również oznaczać brak danych.

3.10 Analiza zmiennej 'wind_dir_avg' - kierunku wiatru wyrażonego w stopniach 0-360

3.10.1 Wykres rozkładu kierunku wiatru

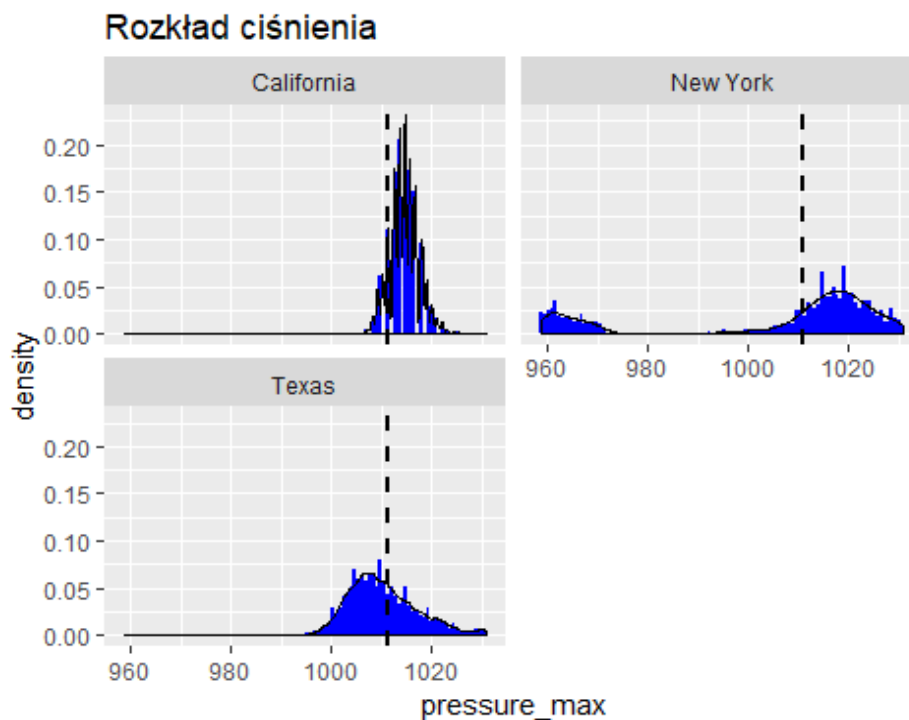


3.10.2 Wnioski:

Każdy ze stanów charakteryzuje się innym rozkładem kierunku wiatru. Ma to ścisły związek z ich położeniem: ocean spokojny, ocean atlantycki, zatoka meksykańska.

3.11 Analiza zmiennej 'pressure_max' - ciśnienia

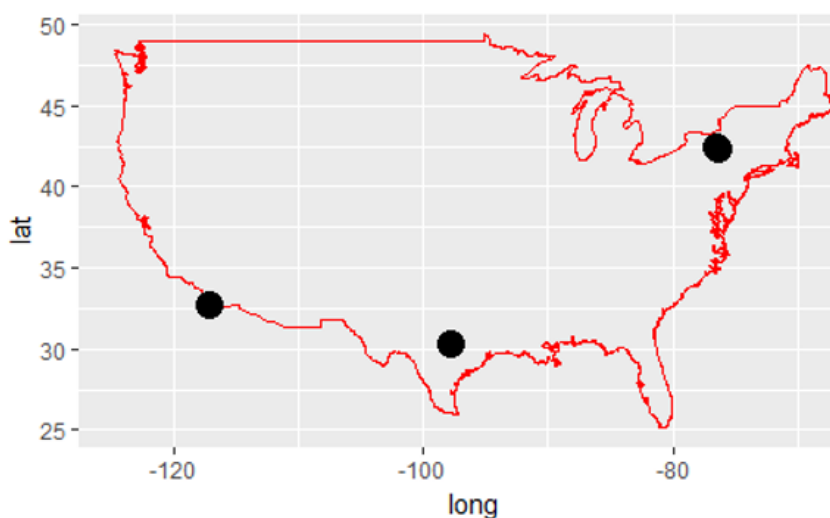
3.11.1 Wykres rozkładu ciśnienia z podziałem na stany



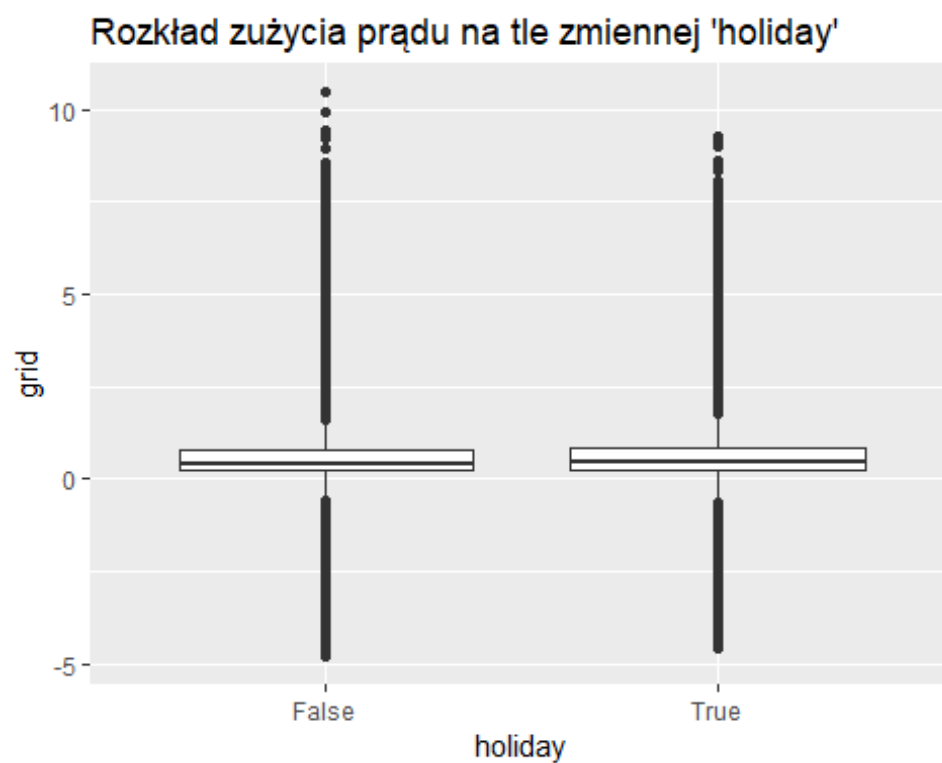
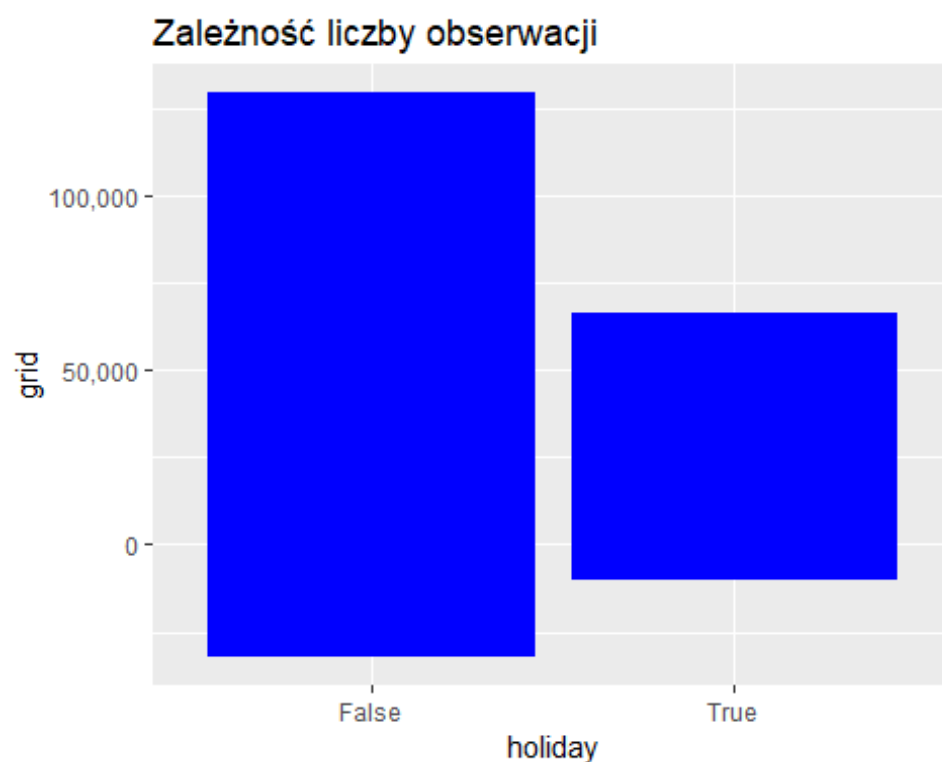
3.11.2 Wnioski:

Silne zróżnicowanie rozkładów ze względu na stan. Szczególnie silne zagęszczenie w przypadku Kalifornii, duże zróżnicowanie w przypadku Nowego Jorku.

3.11.3 Lokalizacje występujące w analizie, na podstawie długości i szerokości geograficznej.



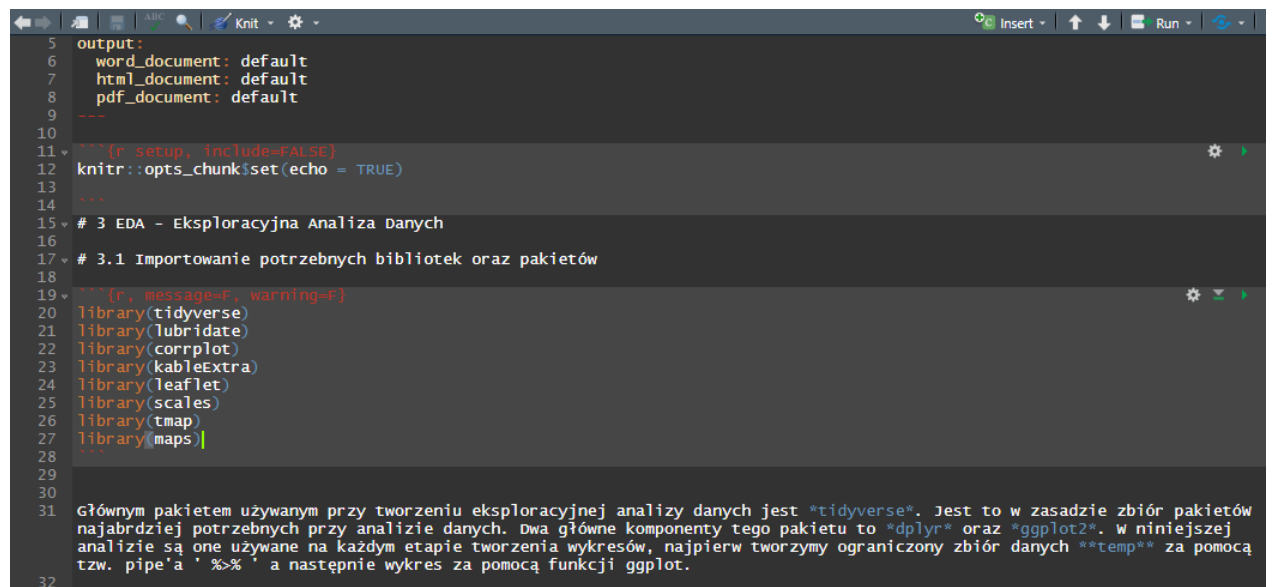
3.11.4 Wykresy zależności zużycia prądu od zmiennej 'holiday', przedstawiające stosunek ilości obserwacji oraz rozkład



3.11.5 Wnioski:

- Porównanie ilościowe, wskazuje na oczywistą różnicę w liczbie obserwacji - więcej jest dni pracujących niż świątecznych.
- W przypadku rozkładu zaprezentowanego na wykresie pudełkowym brak istotnych różnic. Wartości pierwszego i trzeciego kwartyla są przybliżone, tak samo jak wartość mediany.

*Eksploracyjna analiza danych - EDA, została w całości napisana w R Markdown i automatycznie przekonwertowana w dokument typu .docx. Zaletą tego rozwiązania jest możliwość powielania tych samych kroków analizy dla różnych zbiorów danych, bez konieczności tworzenia dokumentu od nowa.



```
5 output:
6   word_document: default
7   html_document: default
8   pdf_document: default
9   ---
10
11   ```{r setup, include=FALSE}
12   knitr::opts_chunk$set(echo = TRUE)
13   ```
14
15   # 3 EDA - Eksploracyjna Analiza Danych
16
17   # 3.1 Importowanie potrzebnych bibliotek oraz pakietów
18
19   ```{r, message=F, warning=F}
20   library(tidyverse)
21   library(lubridate)
22   library(corrplot)
23   library(kableExtra)
24   library(leaflet)
25   library(scales)
26   library(tmap)
27   library(maps)
28   ```
29
30
31   Głównym pakietem używanym przy tworzeniu eksploracyjnej analizy danych jest "tidyverse". Jest to w zasadzie zbiór pakietów
32   najbardziej potrzebnych przy analizie danych. Dwa główne komponenty tego pakietu to "dplyr" oraz "ggplot2". W niniejszej
   analizie są one używane na każdym etapie tworzenia wykresów, najpierw tworzymy ograniczony zbiór danych "temp" za pomocą
   tzw. pipe'a " %>% " a następnie wykres za pomocą funkcji ggplot.
```

Rys. 3.1 Widok programu R Studio, w którym tworzony jest skrypt .rmd

źródło: opracowanie własne

4. Inżynieria cech

W związku z brakiem lub też bardzo słabą korelacją pomiędzy zmienną celu „grid”, a pozostałymi zmiennymi postanowiono przeprowadzić kilka modyfikacji na zbiorze danych, które pozwoliły na stworzenie nowych cech opisujących zmienną celu w lepszym stopniu.

Ze zmiennej ‘local_1hour’, przedstawiającej datę i godzinę w których wykonano pomiar, postanowiono wyciągnąć dwie informacje: godzinę, dzień roku. Zmienne te były lepiej skorelowane ze zmienną celu i opisywały ją lepiej niż zmienna ‘local_1hour’, która jest zmienną typu timestamp (czas uniksowy, czyli liczba sekund, które upłynęły od początku 1970 roku).

```
# Wyciągnięcie zmiennej z godziną pomiaru
df.local_1hour = pd.to_datetime(df.local_1hour)
df['hour'] = df.local_1hour.apply(lambda row: row.hour)
# Wyciągnięcie zmiennej z dniem roku
df['dayofyear'] = df.local_1hour.apply(lambda row: row.dayofyear)
# Usunięcie zmiennej z dokładnym czasem
df = df.drop('local_1hour', axis=1)
```

Jako, że dane zostały zawężone do dwóch miast, postanowiono usunąć zmienne „lat” in „lng” reprezentujące długość i szerokość geograficzną. Tę samą informację niesie zmienna „state” w formacie 0,1 – odpowiednio Teksas, Kalifornia.

```
# Usunięcie szerokości i długości geograficznych - są tylko 2 różne
zestawy # i korespondują ze zmienną 'state'
df = df.drop(['lat', 'lng'], axis=1)
```

Do stworzenia nowych cech opisujących zmienną celu wykorzystano funkcję *PolynomialFeatures* z pakietu Scikit-Learn ze stopniem wielomianu ustawionym na 3. Funkcja ta wygenerowała 170 nowych zmiennych, które powstały przez tworzenie wielomianów 3 stopnia na podstawie cech podanych do funkcji.

Kod wykorzystujący funkcję *PolynomialFeatures*:

```
# Stworzenie nowych zmiennych za pomocą PolynomialFeatures
poly = PolynomialFeatures(degree=3, include_bias=False)
features_to_transform = df[
    ['temp_avg', 'wind_speed_avg', 'wind_dir_avg', 'pressure_max',
     'humidity_avg', 'hour', 'dayofyear',
     'total_square_footage']].copy()
features_transformed = poly.fit_transform(features_to_transform)
col_names = poly.get_feature_names(
    ['temp_avg', 'wind_speed_avg', 'wind_dir_avg', 'pressure_max',
     'humidity_avg', 'hour', 'dayofyear',
     'total_square_footage'])
```

```

new_features = pd.DataFrame(features_transformed, columns=col_names)
new_features = new_features.drop(['temp_avg', 'wind_speed_avg',
'wind_dir_avg', 'pressure_max', 'humidity_avg', 'hour', 'dayofyear',
'total_square_footage'],axis=1)
new_df = pd.concat([df, new_features], axis=1)

```

Tak duża ilość zmiennych nie pozwalała na wystarczająco szybką pracę ze zbiorem, a przede wszystkim nie wszystkie nowe zmienne dobrze opisują zmienną celu. Tym samym należy przystąpić do redukcji wymiarów. Wykorzystując funkcję *SelectKBest* z pakietu Scikit-Learn z testem *f_regression*. Test ten polega na sprawdzeniu indywidualnego efektu każdej z cech na model regresyjny.

Przed przystąpieniem do wyboru najlepszych cech należy przeskalować je do zakresu od 0 do 1. W tym celu użyto funkcji *MinMaxScaler*.

```

# Wybranie najlepszych zmiennych za pomocą funkcji SelectKMax
new_df = new_df.dropna()
df_grid = new_df.grid.copy()
df_grid = df_grid.values.reshape(-1,1)

scaler = MinMaxScaler()
scaler.fit(df_grid)

df_gscaled = scaler.transform(df_grid)
df_numeric = new_df.drop('grid', axis=1).copy()
df_numscaled = scaler.transform(df_numeric)
best_features = SelectKBest(f_regression, k=20).fit(df_numscaled,
df_gscaled)

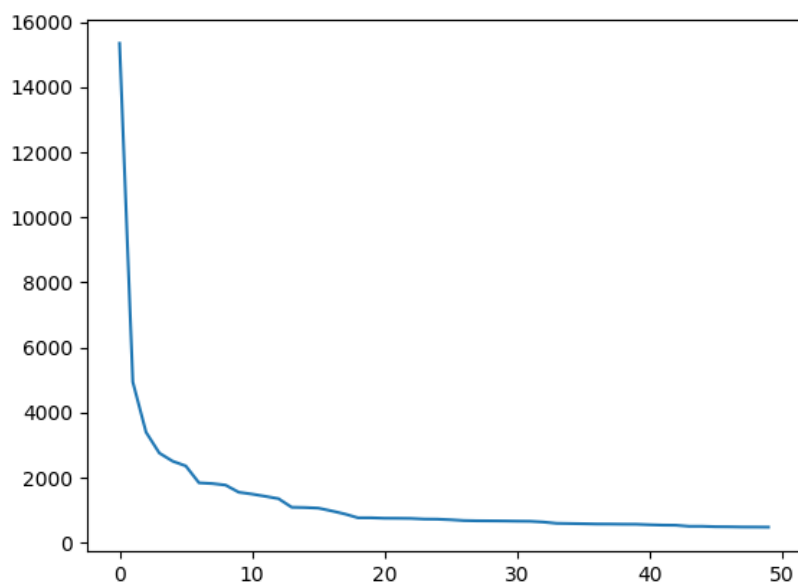
df_scores = pd.DataFrame(best_features.scores_)
df_columns = pd.DataFrame(df_numeric.columns)
feature_scores = pd.concat([df_columns, df_scores], axis=1)
feature_scores.columns = ['Feature', 'Score']

best_features_set = pd.DataFrame()
tmp_cols = df_numeric.columns
df_numscaled = pd.DataFrame(df_numscaled, columns=tmp_cols)

for col in feature_scores.nlargest(20, 'Score')['Feature']:
    best_features_set[col] = df_numeric[col].copy()

tmp_df = pd.concat([best_features_set, pd.DataFrame(df_grid,
columns=['grid',]),], axis=1)

```



Rysunek 6.1 Rozkład wyników funkcji SelectKBest – posortowany od najbardziej znaczących
źródło: opracowanie własne

Jak widać na powyższym rozkładzie, wpływ zmiennych nie zmienia się w sposób znaczący od okolicy 20 pozycji na wykresie. W związku z tym postanowiono wybrać pierwsze 20 najistotniejszych zmiennych.

Feature	Score
humidity_avg	15348.464600
total_square_footage	4940.229396
pressure_max	3395.895127
state	2756.257112
dayofyear^2 total_square_footage	2506.220400
dayofyear total_square_footage^2	2359.281171
pressure_max dayofyear	1839.317512
total_square_footage dayofyear	1817.844815
temp_avg	1771.025022
humidity_avg dayofyear	1550.902009
total_square_footage dayofyear	1494.309708
wind_dir_avg	1425.623919
total_square_footage dayofyear	1353.679006
Apartment	1088.775784
dayofyear^3	1080.327190
pressure_max dayofyear^2	1063.093921
dayofyear^2	978.276571
wind_dir_avg dayofyear^2	883.615680
humidity_avg dayofyear^2	765.649709
dayofyear	764.752279
pressure_max^2	
pressure_max^3	

5. Badanie modeli regresji

W celu zbadania modeli na danych w pierwszej kolejności przystąpiono do podziału danych na zbiory testowy i treningowy w proporcjach 80/20 za pomocą funkcji *train_test_split* z pakietu Scikit-Learn.

```
# Stworzenie zbioru treningowego i testowego
train_set, test_set = train_test_split(data, test_size=0.2,
                                       random_state=42)
```

Następnie ze zbiorów wydzielono zmienną celu.

```
# Stworzenie zestawów cech i etykiet
X_train = train_set.loc[:, : 'pressure_max^3']
y_train = train_set['grid']
X_test = test_set.loc[:, : 'pressure_max^3']
y_test = test_set['grid']
```

Kolejnym krokiem było wytrenowanie modeli i przeprowadzenie ich ewaluacji. W tym celu stworzono funkcję *train_eval_model*, która przeprowadza trening oraz ewaluację modelu i klasę *ResultDataRegressor*, która pozwala na stworzenie obiektu zawierającego wyniki modelu.

```
class ResultDataRegressors:
    """Klasa w której zapisuje wynik każdego algorytmu"""

    def __init__(self, model_name, model,
                 rmse_train, rmse_test, cvs_scores):
        self.model_name = model_name
        self.model = model
        self.rmse_train = rmse_train
        self.rmse_test = rmse_test
        self.cvs_scores = cvs_scores

        self.cvs_mean = self.cvs_scores.mean()
        self.cvs_std = self.cvs_scores.std()

def train_eval_model(model, model_name, X_train, y_train, X_test,
                    y_test):
    """ Funkcja przeprowadza trening i ewaluacje modelu.
        Zwraca obiekt klasy ResultDataRegressor"""
    # Wyniki regresji dla zbioru treningowego
    model_predictions_train = model.predict(X_train)
    # MSE dla zbioru treningowego
    model_mse_train = mean_squared_error(
        y_train, model_predictions_train)
    # RMSE dla zbioru treningowego
    model_rmse_train = np.sqrt(model_mse_train)
    model_predictions_test = model.predict(X_test)
    model_mse_test = mean_squared_error(
        y_test, model_predictions_test)
```

```

model_rmse_test = np.sqrt(model_mse_test)
# Krosvalidacja modelu
model_scores = cross_val_score(model, X_train,
                                y_train, scoring="neg_mean_squared_error", cv=10)
model_rmse_scores = np.sqrt(-model_scores)

model_result = ResultDataRegressors(model_name, model,
                                     model_rmse_train, model_rmse_test,
model_rmse_scores)
return model_result

```

Po tym, jak przygotowano wszystko do trenowania i ewaluacji modelu przyszła pora na podstawienie wybranych modeli w celu wybrania najlepszego dla predykcji zmiennej „grid”. W czasie testów używano modeli z pakietu Scikit-Learn:

- Regresji liniowej (*LinearRegression*)
- Stochastycznego spadku gradientu (*SGDRegressor*)
- Drzewa decyzyjnego (*DecisionTreeRegressor*)
- Maszyn wektorów nośnych (*LinearSVR*)
- Lasu losowego (*RandomForestRegressor*)
- Sieci neuronowej (*MLPRegressor*)

W celu zachowania skalowania wykorzystywanego w poprzednim rozdziale, oraz zachowania możliwości przeprowadzania krosvalidacji w etapie ewaluacji modelu zdecydowano się na użycie funkcji *TransformedTargetRegressor*, która transformuje zmienne przy użyciu wybranego wcześniej *MinMaxScaler*, a następnie rozkodowuje wynik predykcji.

Przykład dla modelu lasu losowego:

```

# Model Lasu Losowego
forest_reg = TransformedTargetRegressor(
    regressor=RandomForestRegressor(
        random_state=42,
        n_jobs=-1,
        n_estimators=250),
    transformer=MinMaxScaler())
forest_reg.fit(X_train, y_train)
forest_result = train_eval_model(
    forest_reg,
    "Random Forest Regressor",
    X_train,
    y_train,
    X_test,
    y_test)

```


6. Wynik i wnioski

Tabela 6.1 Zestawienie wyników końcowych wybranych modeli.

źródło: opracowanie własne

Model	Błąd RMSE trening	Błąd RMSE test	Krosvalidacja	
			Średnia	Odchylenie standardowe
Linear Regressor	4,81E+00	4,77E+00	4,81E+00	3,95E-02
SGD Regressor	6,28E+27	6,23E+27	9,62E+27	5,63E+27
Random Forest Regressor	1,08E+00	2,85E+00	2,94E+00	2,29E-02
LinearSVR	2,23E+01	2,22E+01	2,22E+01	6,07E-02
MLPRegressor	1,33E+07	1,32E+07	5,00E+07	7,02E+07

Jak widać w powyższej tabeli, najlepszy wynik uzyskał model Random Forest Regressor, dla którego RMSE w zbiorze testowym wyniosło 2,85. Niestety, nie jest to wynik satysfakcjonujący biorąc pod uwagę zakres zmiennej celu, który wynosił od -4,7 do 10,5, przy czym większość zbioru zawierało się w zakresie od 0 do 1. Jak zauważono w trakcie eksploracyjnej analizy danych, cechy w zbiorze danych nie wykazywały korelacji ze zmienną celu „grid”. Niestety nie udało się osiągnąć założonego celu jakim była predykcja zużycia energii elektrycznej na podstawie prognozy pogody, gdyż poziom błędu jest na nieakceptowalnym poziomie. W celu poprawy wyników należałoby skupić się na eliminacji wartości odstających, które występują zwłaszcza w zmiennej celu oraz zmiennej „pressure_max”.