1 Разведочный анализ датасета исторических данных по фотоэлектричеству и нагрузке

1.1 Цели и задачи работы

Цели работы:

1. Познакомиться с понятием «большие данные» и способами их обработки;
2. Познакомиться с инструментом Apache Spark и возможностями, которые он предоставляет для обработки больших данных;
3. Получить навыки выполнения разведочного анализа данных использованием PySpark.

Задача работы:

Выполнить разведочный анализ датасета с определением:

* типов признаков в датасете;
* **пропущенных значений и их устранением;**
* выбросов и их устранением;
* **расчетом статистических показателей признаков (средних, квартилей и т.д.);**
* визуализацией распределения наиболее важных признаков;
* корреляций между признаками.

1.2 Описание датасета

Датасет называется “The EMSx dataset: historical photovoltaic and load”. Взят с ресурса: https://www.kaggle.com/datasets/adri1g/the-emsx-dataset-historical-photovoltaic-and-load/data.

Набор данных EMSx – это большая коллекция профилей фотоэлектрических сетей и нагрузок, собранных компанией Schneider Electric с 70 анонимных промышленных объектов. Этот набор данных опубликован как часть бенчмарка EMSx для систем управления энергопотреблением, представленного в родственной публикации DOI: 10.1007/s12667-020-00417-5.

Этот набор данных содержит сжатые файлы с идентификатором сайта в диапазоне от 1 до 70. В этих файлах каждая строка представляет собой временной шаг, в котором собраны исторические данные о производстве фотоэлектричества и потреблении энергии за последние 15 минут, а также исторические прогнозы, рассчитанные компанией Schneider Electric.

В датасете присутствуют следующие столбцы:

* timestamp – время измерения;
* site\_id – номер объекта, с которого брали информацию (0...70);
* period\_id – номер периода;
* actual\_consumption – фактическое потребление энергии;
* actual\_pv – фактическое производство фотоэлектрической энергии;
* load\_00…load 95 – прогноз потребления в кВт/ч на 15 минут вперед вплоть до 24 часов;
* pv\_00…pv\_95 – прогноз производства в кВ/ч на 15 минут вперед вплоть до 24 часов.

1.3 Подготовка данных

Для начала стоит определить, к какому типу относится каждый признак:

* timestamp – признак, содержащий информацию о дате и времени;
* site\_id, period\_id – порядковые признаки, содержащие числовые идентификаторы объекта и периода;
* actual\_consumption, actual\_pv, load\_00…load\_95, pv\_00…pv\_95 – количественные признаки.

Соответственно этой информации преобразуем типы данных в датасете.

Начнем с timestamp. Преобразуем его к типу timestamp с игнорированием часового пояса, так как он везде имеет значение +00:00:

df = df.withColumn("timestamp", df["timestamp"].cast(TimestampType()))

Столбцы site\_id и period\_id преобразуем к типу integer и проверяем их максимальные значения:

#Список столбцов для изменения типа данных на integer

cols\_to\_integer\_conv = ["site\_id", "period\_id"]

#Преобразование типа в int

for col\_name in cols\_to\_integer\_conv:

df = df.withColumn(col\_name, col(col\_name).cast(IntegerType()))

#Проверка максимального значения столбца

max\_site = df.agg(max("site\_id"))

max\_period = df.agg(max("period\_id"))

max\_site.show()

max\_period.show()

Результат:

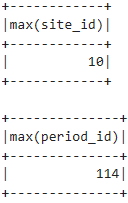


Рисунок 1 – Максимальные значения столбцов site\_id и period\_id

Так как максимумы меньше 127, то мы можем поменять тип на byte, который занимает меньше места в памяти и в теории будет быстрее обрабатываться:

#Так как максимальное значение меньше 127 и весь столбец положителен, то его можно преобразовать в тип byte

for col\_name in cols\_to\_integer\_conv:

df = df.withColumn(col\_name, col(col\_name).cast(ByteType()))

Все количественные признаки преобразуем к типу double:

#Список столбцов для изменения типа данных на double

cols\_to\_double\_conv = ["actual\_consumption", "actual\_pv"] + \

["load\_{:02d}".format(i) for i in range(96)] + \

["pv\_{:02d}".format(i) for i in range(96)]

#Преобразование типа в double

for col\_name in cols\_to\_double\_conv:

df = df.withColumn(col\_name, col(col\_name).cast(DoubleType()))

1.4 Разведочный анализ

1.4.1 Обнаружение и устранение пропущенных значений

Сначала необходимо понять, какое значение для каждого столбца соответствует пропущенному. Например, для period\_id “0” – это действительное значение (номер первого периода), тогда как для всех остальных – отсутствующее.

Далее нужно разобраться, как различные типы данных интерпретируют пропущенные значения. Например, timestamp не может содержать NaN.

Также следует помнить, что в PySpark значения None и null равны.

Суммируя все сказанное, напишем код, который подсчитает количество пропущенных значений в каждом столбце:

null\_values = {}

null\_count = df.where(col("timestamp").isNull()).count()

null\_values.update({"timestamp": null\_count})

null\_count = df.where(col("period\_id").isNull() | isnan(col("period\_id"))).count()

null\_values.update({"period\_id": null\_count})

cols\_to\_check = [col\_name for col\_name in df.columns

if col\_name not in ["timestamp", "period\_id"]]

for column in cols\_to\_check:

null\_count = df.where(col(column).isNull() | isnan(col(column)) | (col(column) == 0)).count()

null\_values.update({column: null\_count})

null\_df = pd.DataFrame.from\_dict([null\_values])

null\_df.iloc[0][null\_df.iloc[0] > 0].T

Результат:

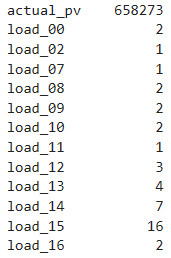


Рисунок 2 – Количество пропущенных значений в столбцах

Так как timestamp и period\_id не имеют пропущенных значений, то мы можем обработать оставшиеся столбцы одним условием на поиск 0, null и NaN:

cols\_to\_check = [column for column in null\_df.columns

if null\_df[column].iloc[0] > 0]

for column in cols\_to\_check:

df = df.where(~(col(column).isin([0, np.nan])) & col(column).isNotNull())

1.4.2 Расчет статистических показаний признаков

Напишем код для вывода статистических данных по столбцам:

summary\_df = df.summary().toPandas()

#Выбираем столбцы, начиная со второго, так как первый - это описание

numeric\_cols = summary\_df.columns[1:]

#Преобразуем к числовому типу

summary\_df[numeric\_cols] = summary\_df[numeric\_cols].apply(pd.to\_numeric)

#Округляем только числовые значения

summary\_df[numeric\_cols] = summary\_df[numeric\_cols].round(2)

summary\_df.T

Результат:

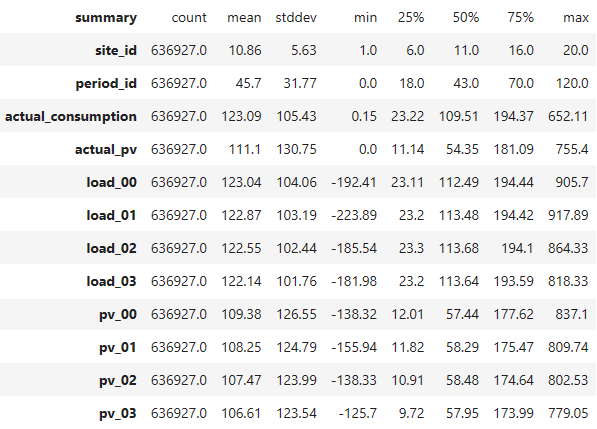


Рисунок 3 – Статистические показания некоторых признаков

По статистическим данным видно, что минимальное значение столбцов load\_xx и pv\_xx может достигать показателя меньше нуля. В контексте их значения – описывать прогноз потребления энергии и производства фотоэлектрических панелей через каждые 15 минут – это недопустимо. Нужно удалить отрицательные значения:

cols\_to\_check = ["pv\_{:02d}".format(i) for i in range(96)] + \

["load\_{:02d}".format(i) for i in range(96)]

1.4.3 Визуализация распределения наиболее важных признаков

Прежде чем заниматься удалением выбросов, нужно проверить распределения признаков для определения метода их устранения.

Распределения actual\_consumption и load\_00...load\_02:

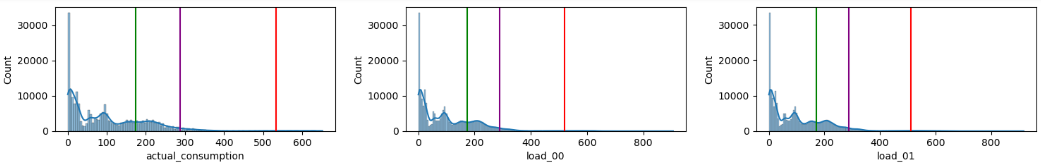


Рисунок 4 – Распределение столбцов actual\_consumption, load\_00 и load\_01

Вывод: распределение этих столбцов близко к экспоненциальному.

Распределения actual\_pv и pv\_00...pv\_02:

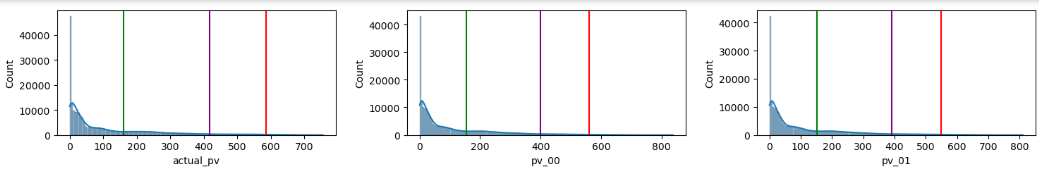


Рисунок 5 – Распределение столбцов actual\_pv, pv\_00 и pv\_01

Вывод: распределение этих столбцов экспоненциальное.

1.4.4 Выбросы и их устранение

В связи с экспоненциальным распределением признаков было принято решение определять выбросы с помощью интерквартильного размаха (IQR). Удаляем значения, выходящие за три IQR от первого и третьего квартилей:

def iqr\_bound(value\_list, num\_iqr=3):

Q1, Q3 = sorted(value\_list)

IQR = Q3 - Q1

lower\_bound = Q1 - num\_iqr \* IQR

upper\_bound = Q3 + num\_iqr \* IQR

return [lower\_bound, upper\_bound]

cols\_to\_check = ["actual\_pv"] + ["actual\_consumption"] + \

["pv\_{:02d}".format(i) for i in range(96)] + \

["load\_{:02d}".format(i) for i in range(96)]

iqr\_df = df

for column in cols\_to\_check:

quantiles = df.approxQuantile(column, [0.25, 0.75], 0.01)

lower\_bound, upper\_bound = iqr\_bound(quantiles)

iqr\_df = iqr\_df.filter((col(column) >= lower\_bound) &

(col(column) <= upper\_bound))

iqr\_pd = iqr\_df.toPandas()

1.4.5 Корреляция признаков

Так как датасет содержит 195 столбцов, целесообразно будет вывести не все корреляции. Выберем самые важные для нас данные и несколько колонок с предсказаниями:

# Извлечение только совпадающих столбцов

target\_columns = ['actual\_consumption', 'actual\_pv', 'load\_00', 'load\_01',

'load\_02', 'load\_03', 'load\_04', 'load\_05', 'load\_06',

'pv\_00', 'pv\_01', 'pv\_02', 'pv\_03', 'pv\_04', 'pv\_05', 'pv\_06']

filtered\_corr\_matrix = corr\_matrix[target\_columns].loc[target\_columns]

# Построение heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(filtered\_corr\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlations')

plt.show();

Результат:

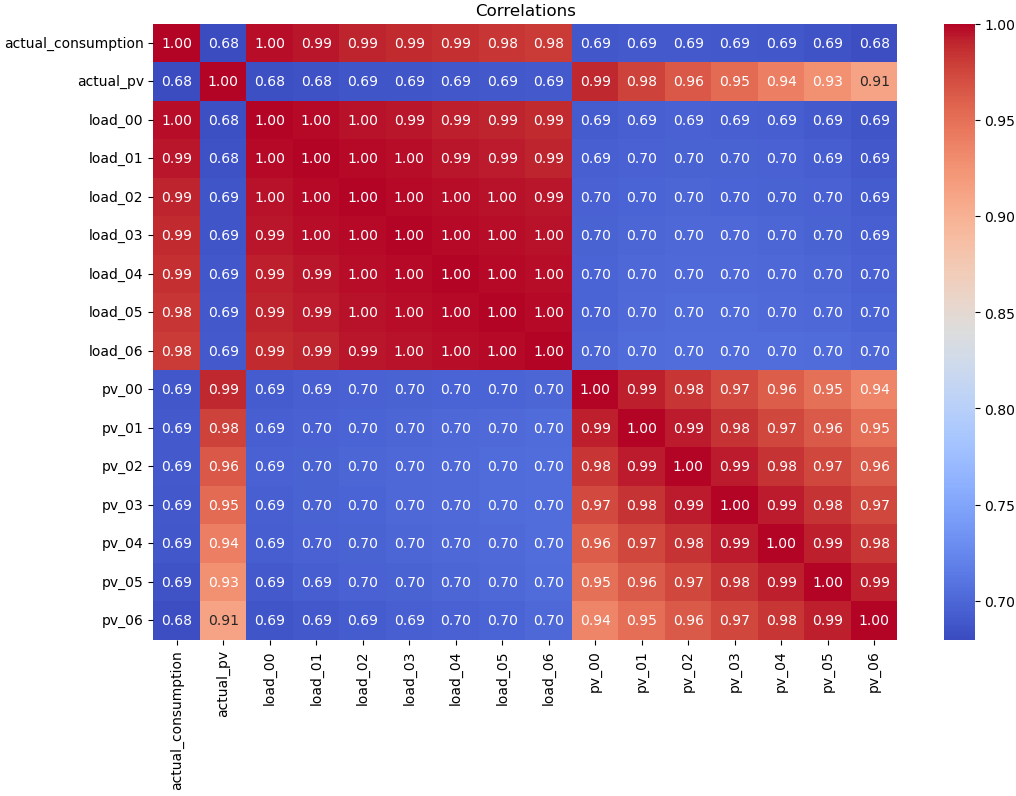


Рисунок 6 – Визуализация корреляции между некоторыми признаками

1.5 Выводы

В этой части курсовой работы был проведен разведочный анализ датасета исторических данных по фотоэлектричеству и нагрузке. Для этого из полного хранилища были выбраны примерно 962 тысячи строк, которые впоследствии сократились примерно до 140 тысяч (с учетом 195 столбцов – это 27300000 ячеек для обработки). Полученные данные имели близкое к экспоненциальному или же экспоненциальное распределение, что затрудняло их аналитику.

Выполненный разведочный анализ заключался в:

* Анализе типов признаков;
* Исправлении типов данных;
* Обнаружении и устранении пропущенных значений;
* Расчете статистических показаний;
* Визуализации распределений;
* Определении и устранении выбросов;
* Просмотре корреляций между признаками.