|  |  |  |
| --- | --- | --- |
| *СИСТЕМЫ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ* | | |
| Лабораторная работа №1  «Разведочный анализ данных с помощью PySpark» | Группа | САПР-1.1 |
| Выполнил | Рублев А.А. |
| Проверил | Кравченя П.Д. |
| Дата выполнения | 29.09.2023 |
| Дата проверки |  |
| Подпись |  |

**Цель и задачи 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.

**Постановка задания**

Согласно варианту №2 «Датасет авиабилетов из Expedia» выполнить разведочный анализ датасета согласно варианту 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.

**Решение**

Датасет содержит информацию о ценах на полеты в одну сторону самолетами согласно Expedia на период с 16.04.2022 до 05.10.2022.

Датасет представляет собой CSV-файл, где каждая строка - купленный билет в/из следующих аэропортов: ATL, DFW, DEN, ORD, LAX, CLT, MIA, JFK, EWR, SFO, DTW, BOS, PHL, LGA, IAD, OAK.

Данные содержат следующую информацию:

* legId: Идентификатор рейса;
* searchDate: Дата (YYYY-MM-DD), когда эта запись была взята из Expedia;
* flightDate: Дата (YYYY-MM-DD) полета;
* startingAirport: Трехзначный код аэропорта IATA для начального местоположения;
* destinationAirport: Трехзначный код аэропорта IATA для места прибытия;
* fareBasisCode: Код базового тарифа;
* travelDuration: Продолжительность поездки в часах и минутах;
* elapsedDays: Количество прошедших дней (обычно 0);
* isBasicEconomy: Логическое значение, указывающее, предназначен ли билет для базовой экономики;
* isRefundable: Логическое значение, указывающее, подлежит ли билет возврату;
* isNonStop: Логическое значение, определяющее, является ли полет беспосадочным;
* baseFare: Цена билета (в долларах США);
* totalFare: Цена билета (в долларах США), включая налоги и другие сборы;
* seatsRemaining: Целое число для количества оставшихся мест;
* totalTravelDistance: Общее расстояние путешествия в милях, иногда эти данные отсутствуют;
* segmentsDepartureTimeEpochSeconds: Строка, содержащая время отправления (время Unix) для каждого этапа поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsDepartureTimeRaw: Строка, содержащая время отправления (формат ISO 8601: ГГГГ-ММ-ДДТчч:мм:сс.000±[чч]:00) для каждого этапа поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsArrivalTimeEpochSeconds: Строка, содержащая время прибытия (время Unix) для каждого этапа поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsArrivalTimeRaw: Строка, содержащая время прибытия (формат ISO 8601: ГГГГ-ММ-ДДТчч:мм:сс.000±[чч]:00) для каждого этапа поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsArrivalAirportCode: Строка, содержащая код аэропорта IATA для места прибытия для каждого этапа поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsDepartureAirportCode: Строка, содержащая код аэропорта IATA для места отправления для каждого этапа поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsAirlineName: Строка, содержащая название авиакомпании, обслуживающей каждый этап поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsAirlineCode: Строка, содержащая двухбуквенный код авиакомпании, обслуживающей каждый этап поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsEquipmentDescription: Строка, содержащая тип самолета, используемого на каждом этапе путешествия (например, «Airbus A321» или «Boeing 737-800»). Записи для каждой из ветвей разделяются знаком «||»;
* segmentsDurationInSeconds: Строка, содержащая продолжительность полета (в секундах) для каждого этапа поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsDistance: Строка, содержащая пройденное расстояние (в милях) для каждого этапа поездки. Записи для каждой из ветвей разделяются знаком «||»;
* segmentsCabinCode: Строка, содержащая салон для каждого этапа поездки (например, «пассажирский»). Записи для каждой из ветвей разделяются знаком «||».

Для упрощения выберем из полного перечня колонок более всего интересующие для анализа: startingAirport, destinationAirport, elapsedDays, isBasicEconomy, isRefundable, isNonStop, baseFare, totalFare, seatsRemaining, totalTravelDistance, segmentsEquipmentDescription.

Данные представляют собой одиночные значения следующих типов данных: int, double, string, boolean. Распределение типов колонок представлено в таблице 1.

pd.DataFrame(df.dtypes, columns = ['Column Name', 'Data type'])

Таблица 1 – Распределение типов данных в колонках

| № | Название колонки | Тип данных |
| --- | --- | --- |
| 1 | startingAirport | string |
| 2 | destinationAirport | string |
| 3 | elapsedDays | int |
| 4 | isBasicEconomy | boolean |
| 5 | isRefundable | boolean |
| 6 | isNonStop | boolean |
| 7 | baseFare | double |
| 8 | totalFare | double |
| 9 | seatsRemaining | int |
| 10 | totalTravelDistance | int |
| 11 | segmentsEquipmentDescription | string |

Всего в датасете имеется 5.999.739 строк, но для уменьшения времени расчетов сократим размер в 6 раз, оставив в нем 999998 строк. Так как датасет представляет из себя csv-файл, загрузку и обрезку осуществим при помощи программы Neovim.

После получения первого представления о колонках, в первую очередь необходимо убедиться, имеет ли датасет какие-либо пропущенные и аномальные нулевые значения.

* Для строковых колонок проверим на None и null;
* Для числовых колонок, допускающих значения ноль, проверим на None и NaN;
* Для числовых колонок, недопускающих значения ноль, проверим на нули, None и NaN;
* Для булевых колонок проверим на None и null.

string\_columns = [

'startingAirport', 'destinationAirport', 'segmentsEquipmentDescription'

]

numeric\_with\_zeroes\_columns = ['elapsedDays', 'seatsRemaining']

numeric\_without\_zeroes\_columns = ['baseFare', 'totalFare', 'totalTravelDistance']

boolean\_columns = ['isBasicEconomy', 'isRefundable', 'isNonStop']

missing\_values = {}

for index, column in enumerate(df.columns):

if column in string\_columns: # check None and Null

missing\_count = df.filter(col(column).eqNullSafe(None) | col(column).isNull()).count()

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

if column in numeric\_with\_zeroes\_columns: # check None, NaN and Null

missing\_count = df.filter(col(column) == None | isnan(col(column)) | col(column).isNull()).count()

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

if column in numeric\_without\_zeroes\_columns: # check zeroes, None, NaN and Null

missing\_count = df.filter(col(column).isin([0,None]) | isnan(col(column)) | col(column).isNull()).count()

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

if column in boolean\_columns: # check None and Null

missing\_count = df.filter(col(column).eqNullSafe(None) | col(column).isNull()).count()

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

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

columns\_with\_missing\_values = []

for column in missing\_df:

if missing\_df[column].values[0] != 0:

columns\_with\_missing\_values.append(column)

missing\_df[columns\_with\_missing\_values]

Количество удаленных записей за каждый из типов колонок представлен в таблице 2.

Таблица 2 – Удаленное количество строк за каждую колонку

|  |  |  |
| --- | --- | --- |
| № | totalTravelDistance | segmentsEquipmentDescription |
| 0 | 61860 | 18754 |

df = df.na.drop(subset=columns\_with\_missing\_values)

print(f'Number of rows after deleting na values: {df.count()}')

Количество строк после удаления пропущенных и аномальных нулевых значений – 921949.

Перед обработкой данных сперва взглянем на статистические показатели числовых признаков, чтобы иметь представление о том, какие данные можно считать аномальными.

Рассчитаем такие показатели, как:

* Минимальное значение;
* Среднее значение;
* Максимальное значение;
* Среднеквадратичное отклонение.

def calculate\_statistical\_indicators(df, column):

min\_value = df.agg(col\_min(column).alias(f'min\_{column}')).collect()[0][f'min\_{column}']

mean\_value = df.agg(col\_mean(column).alias(f'mean\_{column}')).collect()[0][f'mean\_{column}']

max\_value = df.agg(col\_max(column).alias(f'max\_{column}')).collect()[0][f'max\_{column}']

stddev\_value = df.agg(stddev(column).alias(f'stddev\_{column}')).collect()[0][f'stddev\_{column}']

return (min\_value, mean\_value, max\_value, stddev\_value)

statistical\_indicators = {}

for column in numeric\_with\_zeroes\_columns:

statistical\_indicators[column] = calculate\_statistical\_indicators(df, column)

for column in numeric\_without\_zeroes\_columns:

statistical\_indicators[column] = calculate\_statistical\_indicators(df, column)

for key, value in statistical\_indicators.items():

print(f"""

Column: {key}

-- Min: {value[0]}

-- Mean: {value[1]}

-- Max: {value[2]}

-- Stddev: {value[3]}

""")

Результаты поиска статистических показателей числовых признаков показан на рисунке 1.

A screenshot of a computer code

Description automatically generated

Рисунок 1 – Статистические показатели числовых признаков

Построим графики Quantile-Quantile Plot, чтобы увидеть распределение данных. Графики числовых признаков показаны на рисунках 2-6.

def quantile\_quantile\_plot(df, column):

rdd\_map = df.select(column).rdd.map(lambda x: (x, 1))

rdd\_reduced = rdd\_map.reduceByKey(lambda x, y: x + y)

tupled\_data = rdd\_reduced.collect()

data = [tuple[0][column] for tuple in tupled\_data for i in range(tuple[1])]

scipy\_stats.probplot(data, dist='norm', plot=pylab)

pylab.show()

for column in numeric\_with\_zeroes\_columns:

print(f'Column: {column}')

quantile\_quantile\_plot(df, column)

for column in numeric\_without\_zeroes\_columns:

print(f'Column: {column}')

quantile\_quantile\_plot(df, column)

A graph with lines and a red line

Description automatically generated

Рисунок 2 – Распределение данных колонки «elapsedDays»

A graph with blue lines and red lines

Description automatically generated

Рисунок 3 – Распределение данных колонки «seatsRemaining»

A graph with a line and a red line

Description automatically generated

Рисунок 4 – Распределение данных колонки «baseFare»

A graph with a line drawn on it

Description automatically generated

Рисунок 5 – Распределение данных колонки «totalFare»

A graph with a line going up

Description automatically generated

Рисунок 6 – Распределение данных колонки «totalTravelDistance»

Построим круговую диаграмму распределения двух категориальных признаков: аэропорт начального местоположения и аэропорт места прибытия. Круговые диаграммы представлены на рисунках 7-8.

def pie\_chart(df, column):

pandas\_df = df.groupby(column).count().toPandas()

pandas\_df = pandas\_df.set\_index(pandas\_df.columns[0])

fig, ax = plt.subplots(figsize=(12, 7), subplot\_kw=dict(aspect='equal'), dpi=120)

data = pandas\_df['count']

categories = pandas\_df.index

def func(pct, allvals):

absolute = int(pct/100. \* np.sum(allvals))

return f'{pct:.1f}% ({absolute:d})'

wedges, texts, autotexts = ax.pie(

data,

autopct=lambda pct: func(pct, data),

textprops=dict(color='w'),

colors=plt.cm.Dark2.colors,

startangle=140

)

ax.legend(wedges, categories, title='Airport code', loc='center left', bbox\_to\_anchor=(1, 0, 0.5, 1))

plt.setp(autotexts, size=6, weight=700)

ax.set\_title(f'{column}: Pie Chart')

plt.show()

airports\_columns = ['startingAirport', 'destinationAirport']

for column in airports\_columns:

print(f'Column: {column}')

pie\_chart(df, column)

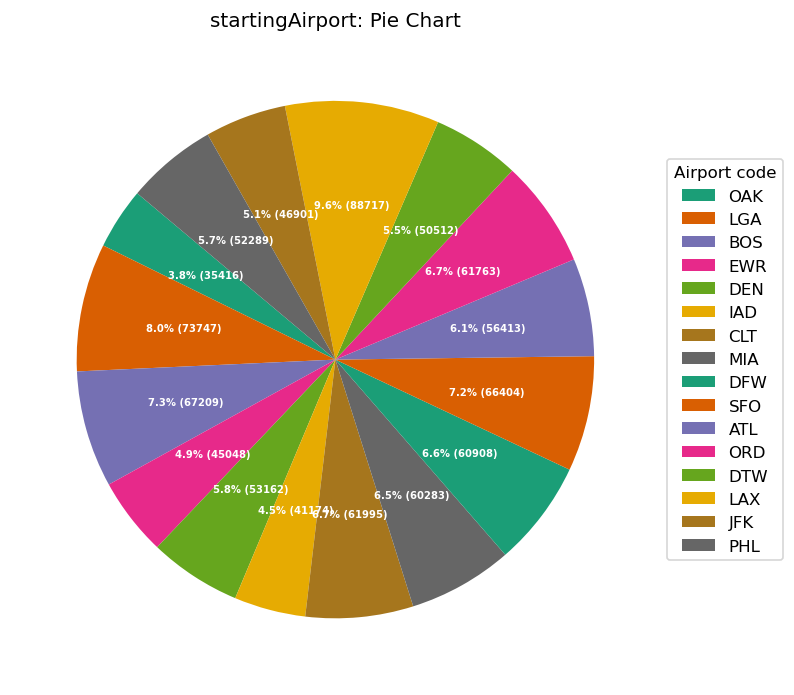


Рисунок 7 – Распределение данных колонки «startingAirport»

A colorful pie chart with numbers and letters

Description automatically generated

Рисунок 8 – Распределение данных колонки «destinationAirport»

Прежде чем удалить выбросы, необходимо сначала решить, что считается выбросом. Есть два распространенных способа сделать это:

1. Использование межквартильного диапазона

Межквартильный размах () — это разница между 75-м процентилем () и 25-м процентилем () в наборе данных. Он измеряет разброс средних 50% значений.

Наблюдение можно определить как выброс, если оно в 1,5 раза превышает межквартильный размах, превышающий третий квартиль (), или в 1,5 раза превышает межквартильный размах, меньше первого квартиля ().

2. Использование z-значения

Z-оценка показывает, сколько стандартных отклонений данного значения от среднего. Используем следующую формулу для расчета z-показателя:

где:

* *X* - это одно необработанное значение данных;
* *μ* - среднее значение населения;
* *σ* - стандартное отклонение населения.

Можно определить наблюдение как выброс, если его z-оценка меньше -3 или больше 3.

Результаты работы методов Z-оценки и межквартильного диапазона представлены на рисунках 9-10.

def z\_estimation(df, column):

z\_values\_column\_name = 'z'

row\_idx\_column\_name = 'row\_idx'

data = np.array(df.select(collect\_list(column)).first()[0])

z = np.abs(scipy\_stats.zscore(data))

df\_clean = spark.createDataFrame(z, [z\_values\_column\_name])

df = df.withColumn(row\_idx\_column\_name, row\_number().over(Window.orderBy(monotonically\_increasing\_id())))

df\_clean = df\_clean.withColumn(row\_idx\_column\_name, row\_number().over(Window.orderBy(monotonically\_increasing\_id())))

df\_clean = df.join(df\_clean, df.row\_idx == df\_clean.row\_idx)

return df\_clean.filter(col(z\_values\_column\_name) < 3).drop(row\_idx\_column\_name, z\_values\_column\_name)

z = {}

df\_z = df

for column in numeric\_with\_zeroes\_columns:

df\_z = z\_estimation(df\_z, column)

print(f"""

Column: {column}

-- Cleaned: {df.count()} - {df\_z.count()} = {df.count() - df\_z.count()}

""")

for column in numeric\_without\_zeroes\_columns:

df\_z = z\_estimation(df\_z, column)

print(f"""

Column: {column}

-- Cleaned: {df.count()} - {df\_z.count()} = {df.count() - df\_z.count()}

""")

print(f'Total cleaned: {df.count()} - {df\_z.count()} = {df.count() - df\_z.count()}')

def interquantile\_range(df, column):

quantileProbabilities = [0.25, 0.5, 0.75]

quantiles = df.approxQuantile(column, quantileProbabilities, 0.01)

quantile25th = quantiles[0]

median = quantiles[1]

quantile75th = quantiles[2]

data = np.array(df.select(collect\_list(column)).first()[0])

iqr = np.array(scipy\_stats.iqr(data))

df\_clean = df.filter(~((col(column) < (quantile25th - 1.5\*iqr)) | (col(column) > (quantile75th + 1.5\*iqr))))

return (df\_clean, quantile25th, median, quantile75th, iqr)

quantiles = {}

df\_quantiles = df

for column in numeric\_with\_zeroes\_columns:

quantiles\_info = interquantile\_range(df\_quantiles, column)

df\_quantiles = quantiles\_info[0]

quantiles[column] = quantiles\_info[1:]

print(f"""

Column: {column}

-- Cleaned: {df.count()} - {df\_quantiles.count()} = {df.count() - df\_quantiles.count()}

""")

for column in numeric\_without\_zeroes\_columns:

quantiles\_info = interquantile\_range(df\_quantiles, column)

df\_quantiles = quantiles\_info[0]

quantiles[column] = quantiles\_info[1:]

print(f"""

Column: {column}

-- Cleaned: {df.count()} - {df\_quantiles.count()} = {df.count() - df\_quantiles.count()}

""")

for key, value in quantiles.items():

print(f"""

Column: {key}

-- Quantile 25th: {value[0]}

-- Median: {value[1]}

-- Quantile 75th: {value[2]}

-- IQR: {value[3]}

""")

print(f'Total cleaned: {df.count()} - {df\_quantiles.count()} = {df.count() - df\_quantiles.count()}')

A screenshot of a computer program

Description automatically generated

Рисунок 9 – Результаты работы метода Z-оценки

A screenshot of a computer

Description automatically generated

Рисунок 10 – Результаты работы метода межквартильного диапазона

Как можно увидеть, метод межквартильного диапазона убирает довольно большую часть данных, так как граница в 75%, выше которой данные считаются аномальными, в случае, например, с elapsedDays, является довольно грубой. Конкретно с текущим датасетом использование данного метода не является наилучшим.

Метод Z-оценки является более оптимальным, поскольку обнаруживает действительно слишком отличающиеся значения.

Чтобы выявить связи между признаками, можно построить матрицу корреляций для числовых признаков. Матрица корреляций показана на рисунке 11.

vector\_col = 'corr\_features'

numeric\_columns = numeric\_with\_zeroes\_columns + numeric\_without\_zeroes\_columns

assembler = VectorAssembler(inputCols=numeric\_columns, outputCol=vector\_col)

df\_vector = assembler.transform(df\_z).select(vector\_col)

matrix = Correlation.corr(df\_vector, vector\_col).collect()[0][0]

corr\_matrix = matrix.toArray().tolist()

corr\_matrix\_df = pd.DataFrame(data=corr\_matrix, columns=numeric\_columns, index=numeric\_columns)

plt.figure(figsize=(16,5))

sns.heatmap(

corr\_matrix\_df,

xticklabels=corr\_matrix\_df.columns.values,

yticklabels=corr\_matrix\_df.columns.values,

cmap='Greens',

annot=True

)

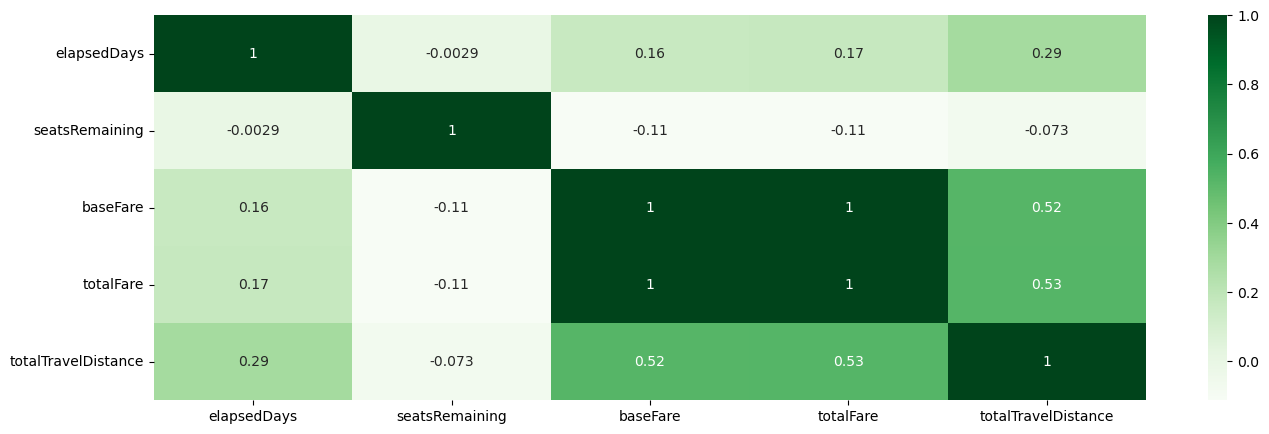


Рисунок 11 – Корреляционная матрица

Как видно из корреляционной матрицы:

* baseFare и totalFare имеют сильную положительную корреляцию, поскольку, очевидно, что на одну и ту же цену билета всегда будет одна и та же сумма налогов;
* baseFare, totalFare и totalTravelDistance имеют среднюю положительную корреляцию, так как цена билета в основном зависит от общего пройденного расстояния;
* линейная корреляция между остальными столбцами практически отсутствует, так как блика к нулю.

**Выводы**

В ходе лабораторной работы был проведен разведочный анализ данных с помощью PySpark: определены типы признаков в датасете, устранены пропущенные значения и выбросы, рассчитаны статистические показатели признаков, визуализированы распределение наиболее важных признаков и корреляция между ними.