МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа №2**

по дисциплине«Методы машинного обучения»

Тема: «Обработка признаков (часть 1)»  
Вариант 9

ИСПОЛНИТЕЛЬ: \_\_\_Очеретная С.В.\_\_

ФИО

группа ИУ5-25М \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2024 г.

ПРЕПОДАВАТЕЛЬ: \_\_\_\_\_Гапанюк Ю.Е.\_\_\_\_

ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2024 г.

Москва - 2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Цель лабораторной работы:** изучение продвинутых способов предварительной обработки данных для дальнейшего формирования моделей.

# Задание

1. Выбрать набор данных (датасет), содержащий категориальные и числовые признаки и пропуски в данных. Для выполнения следующих пунктов можно использовать несколько различных наборов данных (один для обработки пропусков, другой для категориальных признаков и т.д.) Просьба не использовать датасет, на котором данная задача решалась в лекции.
2. Для выбранного датасета (датасетов) на основе материалов лекций решить следующие задачи:

* устранение пропусков в данных;
* кодирование категориальных признаков;
* нормализация числовых признаков.

# Ход работы

## Импорт данных и библиотек

Импорт библиотек

import numpy as np  
import pandas as pd  
import matplotlib as mpl  
import matplotlib.pyplot as plt  
%matplotlib inline  
import seaborn as sns  
from sklearn.impute import SimpleImputer  
from sklearn.impute import MissingIndicator  
from category\_encoders.count import CountEncoder as ce\_CountEncoder  
from category\_encoders.target\_encoder import TargetEncoder as ce\_TargetEncoder  
from category\_encoders.woe import WOEEncoder as ce\_WOEEncoder  
import scipy.stats as stats  
!pip install category\_encoders

Подключение к диску

from google.colab import drive

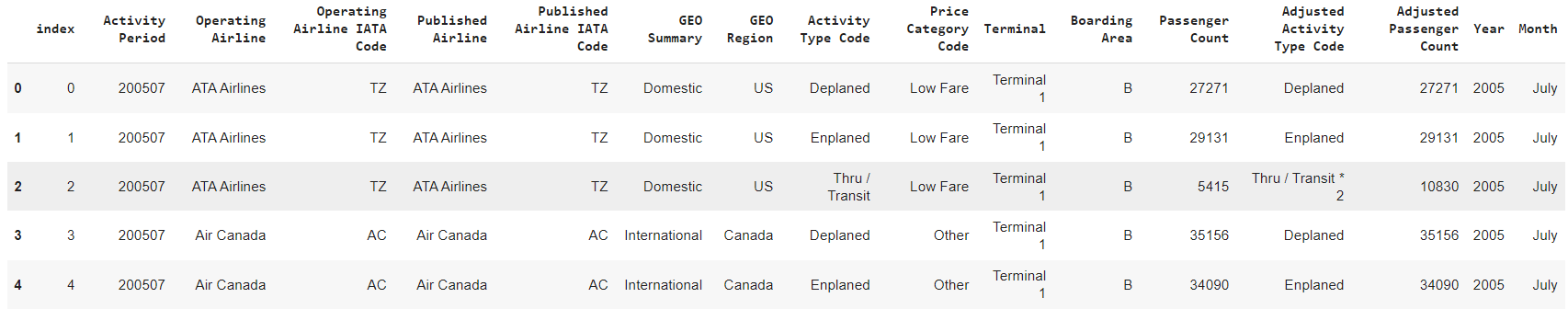
Импорт датасета

data = pd.read\_csv("/content/drive/My Drive/Учеба/магистратура/2 сем/ММО/Air\_Traffic\_Passenger\_Statistics.csv")  
data2 = pd.read\_csv("/content/drive/My Drive/Учеба/магистратура/2 сем/ММО/ryanair\_reviews.csv")

## Описание датасета Air\_Traffic\_Passenger\_Statistics

Просмотр первых 5 строк датасета

data.head()



data.shape

(15007, 17)

Датасет содержит информацию о статистике пассажиропотока авиакомпании. Статистика пассажиропотока может быть полезным инструментом для понимания авиационной отрасли и составления планов поездок. Данные включают количество пассажиров, выполняющую авиакомпанию, опубликованную авиакомпанию, географический регион, код типа деятельности, код ценовой категории, терминал, место посадки, а также год и месяц полета

Датасет состоит из 17 колонок:

1. index - номер колонки
2. Activity Period - период активности (Date)
3. Operating Airline - авиакомпания, выполнявшая рейс(String)
4. Operating Airline IATA Code - IATA-код авиакомпании, выполнявшей рейс (String)
5. Published Airline - авиакомпания, опубликовавшая тариф на рейс (String)
6. Published Airline - IATA-код авиакомпании, опубликовавшей тариф на рейс (String)
7. GEO Summary - краткая информация о географическом регионе (String)
8. GEO Region T - географический регион (String)
9. Activity Type Code - вид деятельности (String)
10. Price Category Code - ценовая категория тарифа (String)
11. Terminal - терминал рейса (String)
12. Boarding Area - зона посадки на рейс (String)
13. Passenger Count - количество пассажиров на рейсе (Integer)
14. Adjusted Activity Type Code - вид деятельности, скорректированный с учетом отсутствующих данных (String)
15. Adjusted Passenger Count - количество пассажиров на рейсе, скорректированное с учетом отсутствующих данных (Integer)
16. Year - год активности (Integer)
17. Month - месяц активности (Integer)

Уникальные значения некоторых колонок:

print(data['GEO Summary'].unique())  
print(data['GEO Region'].unique())  
print(data['Activity Type Code'].unique())  
print(data['Price Category Code'].unique())  
print(data['Terminal'].unique())  
print(data['Boarding Area'].unique())  
print(data['Adjusted Activity Type Code'].unique())

['Domestic' 'International']  
['US' 'Canada' 'Asia' 'Europe' 'Australia / Oceania' 'Mexico'  
 'Central America' 'Middle East' 'South America']  
['Deplaned' 'Enplaned' 'Thru / Transit']  
['Low Fare' 'Other']  
['Terminal 1' 'International' 'Terminal 3' 'Other' 'Terminal 2']  
['B' 'G' 'A' 'E' 'C' 'F' 'Other' 'D']  
['Deplaned' 'Enplaned' 'Thru / Transit \* 2']

Нулевые значения всего в 2х колонках, но для анализа мы их использовать не будем. Вырежем колонки, которые не будем использовать:

data=data.drop(columns=['index','Operating Airline IATA Code','Published Airline IATA Code'],axis=1)

data.isnull().sum().sum()

0

list(zip(data.columns, [i for i in data.dtypes]))

[('Activity Period', dtype('int64')),  
 ('Operating Airline', dtype('O')),  
 ('Published Airline', dtype('O')),  
 ('GEO Summary', dtype('O')),  
 ('GEO Region', dtype('O')),  
 ('Activity Type Code', dtype('O')),  
 ('Price Category Code', dtype('O')),  
 ('Terminal', dtype('O')),  
 ('Boarding Area', dtype('O')),  
 ('Passenger Count', dtype('int64')),  
 ('Adjusted Activity Type Code', dtype('O')),  
 ('Adjusted Passenger Count', dtype('int64')),  
 ('Year', dtype('int64')),  
 ('Month', dtype('O'))]

## Описание датасета ryanair\_reviews

Этот набор данных объединяет разнообразные мнения и рейтинги, предоставленные непосредственно самими пассажирами и охватывающие различные аспекты рейсов Ryanair с 2012 по 2024 год.

В этом наборе данных можно найти:

* Оценки пассажиров по комфорту сидений, обслуживанию бортпроводников, еде и напиткам, наземному обслуживанию и общему соотношению цены и качества.
* Подробная информация о типах путешественников, таких как туристические, деловые или семейные.
* Информация о типах самолетов, типах мест, маршрутах и датах путешествия.
* Разбивка по национальностям пассажиров и статусу подтверждения поездки.

data2.shape

(2249, 21)

Типы колонок:

list(zip(data2.columns, [i for i in data2.dtypes]))

[('Unnamed: 0', dtype('int64')),  
 ('Date Published', dtype('O')),  
 ('Overall Rating', dtype('float64')),  
 ('Passenger Country', dtype('O')),  
 ('Trip\_verified', dtype('O')),  
 ('Comment title', dtype('O')),  
 ('Comment', dtype('O')),  
 ('Aircraft', dtype('O')),  
 ('Type Of Traveller', dtype('O')),  
 ('Seat Type', dtype('O')),  
 ('Origin', dtype('O')),  
 ('Destination', dtype('O')),  
 ('Date Flown', dtype('O')),  
 ('Seat Comfort', dtype('float64')),  
 ('Cabin Staff Service', dtype('float64')),  
 ('Food & Beverages', dtype('float64')),  
 ('Ground Service', dtype('float64')),  
 ('Value For Money', dtype('float64')),  
 ('Recommended', dtype('O')),  
 ('Inflight Entertainment', dtype('float64')),  
 ('Wifi & Connectivity', dtype('float64'))]

### Исправление "багов" датасета

Некоторые колонки с категориальными признаками имеют значения, одинаковые по смыслу, но разные по написанию. Например "NotVerified" тоже самое, что и "Not Verified". Исправим это.

data2['Trip\_verified'] = data2['Trip\_verified'].replace('Not Verified','NotVerified')  
data2['Trip\_verified'] = data2['Trip\_verified'].replace('Unverified','NotVerified')

## Устранение пропусков

Устранять пропуски будем для датасета ryanair\_reviews

### Поиск пропусков

Колонки с пропусками

cols\_with\_na = [c for c in data2.columns if data2[c].isnull().sum() > 0]  
cols\_with\_na

['Overall Rating',  
 'Trip\_verified',  
 'Aircraft',  
 'Type Of Traveller',  
 'Origin',  
 'Destination',  
 'Date Flown',  
 'Seat Comfort',  
 'Cabin Staff Service',  
 'Food & Beverages',  
 'Ground Service',  
 'Value For Money',  
 'Inflight Entertainment',  
 'Wifi & Connectivity']

Числовые колонки с пропусками

number\_cols\_with\_na = [c for c in data2.columns if (data2[c].isnull().sum() > 0 and data2[c].dtype == 'float64')]  
number\_cols\_with\_na

['Overall Rating',  
 'Seat Comfort',  
 'Cabin Staff Service',  
 'Food & Beverages',  
 'Ground Service',  
 'Value For Money',  
 'Inflight Entertainment',  
 'Wifi & Connectivity']

Категориальные колонки с пропусками

cat\_cols\_with\_na = [c for c in data2.columns if (data2[c].isnull().sum() > 0 and data2[c].dtype == 'O')]  
cat\_cols\_with\_na

['Trip\_verified',  
 'Aircraft',  
 'Type Of Traveller',  
 'Origin',  
 'Destination',  
 'Date Flown']

Колличество пропусков

data2.isnull().sum()

Unnamed: 0 0  
Date Published 0  
Overall Rating 130  
Passenger Country 0  
Trip\_verified 944  
Comment title 0  
Comment 0  
Aircraft 1698  
Type Of Traveller 614  
Seat Type 0  
Origin 615  
Destination 615  
Date Flown 618  
Seat Comfort 112  
Cabin Staff Service 121  
Food & Beverages 937  
Ground Service 671  
Value For Money 1  
Recommended 0  
Inflight Entertainment 1918  
Wifi & Connectivity 1981  
dtype: int64

Доли пропусков

[(c, data2[c].isnull().mean()) for c in cols\_with\_na]

[('Overall Rating', 0.057803468208092484),  
 ('Trip\_verified', 0.41974210760337927),  
 ('Aircraft', 0.7550022232103157),  
 ('Type Of Traveller', 0.2730102267674522),  
 ('Origin', 0.2734548688305914),  
 ('Destination', 0.2734548688305914),  
 ('Date Flown', 0.2747887950200089),  
 ('Seat Comfort', 0.04979991107158737),  
 ('Cabin Staff Service', 0.05380168963983993),  
 ('Food & Beverages', 0.41662961316140507),  
 ('Ground Service', 0.29835482436638505),  
 ('Value For Money', 0.00044464206313917296),  
 ('Inflight Entertainment', 0.8528234771009338),  
 ('Wifi & Connectivity', 0.8808359270787016)]

### Удаление колонок с пропусками

Для примера удалим нулевые значения для 2х колонок и просмотрим изменения

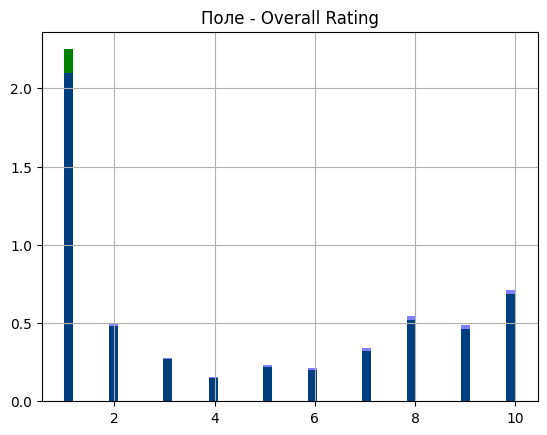
cols2\_with\_na = ['Overall Rating', 'Seat Comfort']

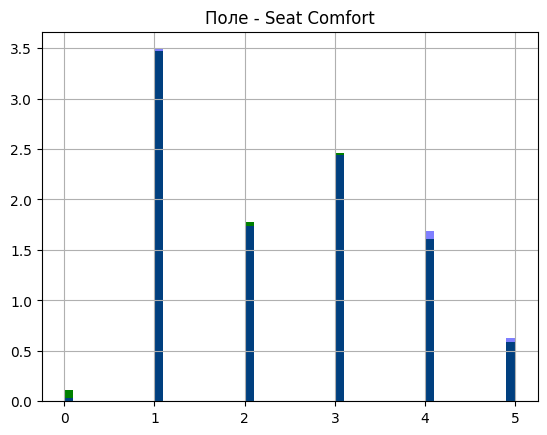
data\_droped = data2[cols2\_with\_na].dropna()

Распределение данных до и после пропусков

def plot\_hist\_diff(old\_ds, new\_ds, cols):  
 for c in cols:  
 fig = plt.figure()  
 ax = fig.add\_subplot(111)  
 ax.title.set\_text('Поле - ' + str(c))  
 old\_ds[c].hist(bins=50, ax=ax, density=True, color='green')  
 new\_ds[c].hist(bins=50, ax=ax, color='blue', density=True, alpha=0.5)  
 plt.show()

plot\_hist\_diff(data2, data\_droped, cols2\_with\_na)





Как видим, после перераспределения, значений с рейтингом 1 стало чуть меньше. Значений остальных оценок стало чуть больше. При этом, если удалять не для 2 колонок, а для большего числа, очень много реальных значений пропадут (т.к. если есть нулевое значение в строке какой-то колонки - строка удалится, но при этом значений других колонок в этой строке просто пропадут). Отсюда следует вывод о том, что надо не удалять строки, а просто замещать пустые значения в них

### Заполнение пропусков

#### Числовые признаки

**Среднее, медиана, мода**

Для внедрения значений будем использовать класс SimpleImputer, для фильтрации пропущенных значений - MissingIndicator.

Пример заполнения пропусков средним для 1го столбца

col\_to\_fill = number\_cols\_with\_na[0]; # 0 -> 'Overall Rating'  
col\_to\_fill

{"type":"string"}

def impute\_column(dataset, column, strategy\_param, fill\_value\_param=None):  
 """  
 Заполнение пропусков в одном признаке  
 """  
 temp\_data = dataset[[column]].values  
 size = temp\_data.shape[0]  
  
 indicator = MissingIndicator()  
 mask\_missing\_values\_only = indicator.fit\_transform(temp\_data)  
  
 imputer = SimpleImputer(strategy=strategy\_param,  
 fill\_value=fill\_value\_param)  
 all\_data = imputer.fit\_transform(temp\_data)  
  
 missed\_data = temp\_data[mask\_missing\_values\_only]  
 filled\_data = all\_data[mask\_missing\_values\_only]  
  
 return all\_data.reshape((size,)), filled\_data, missed\_data

Заполним столбец средним значением и просмотрим это среднее значение

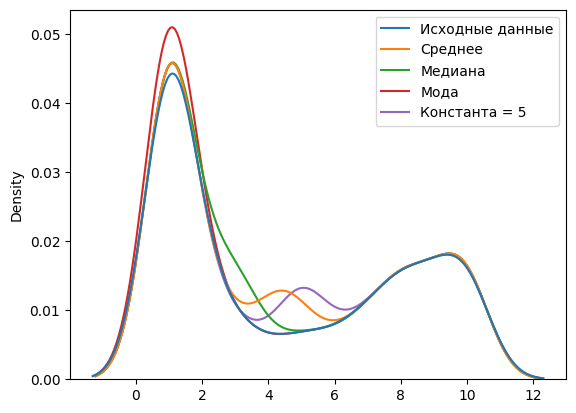
all\_data, filled\_data, missed\_data = impute\_column(data2, col\_to\_fill, 'mean')  
filled\_data[0]

4.380368098159509

Просмотрим распределение разных метрик для заполнения пропусков

def research\_impute\_numeric\_column(dataset, num\_column, const\_value=None):  
 strategy\_params = ['mean', 'median', 'most\_frequent', 'constant']  
 strategy\_params\_names = ['Среднее', 'Медиана', 'Мода']  
 strategy\_params\_names.append('Константа = ' + str(const\_value))  
  
 original\_temp\_data = dataset[[num\_column]].values  
 size = original\_temp\_data.shape[0]  
 original\_data = original\_temp\_data.reshape((size,))  
  
 new\_df = pd.DataFrame({'Исходные данные':original\_data})  
  
 for i in range(len(strategy\_params)):  
 strategy = strategy\_params[i]  
 col\_name = strategy\_params\_names[i]  
 if (strategy!='constant') or (strategy == 'constant' and const\_value!=None):  
 if strategy == 'constant':  
 temp\_data, \_, \_ = impute\_column(dataset, num\_column, strategy, fill\_value\_param=const\_value)  
 else:  
 temp\_data, \_, \_ = impute\_column(dataset, num\_column, strategy)  
 new\_df[col\_name] = temp\_data  
  
 sns.kdeplot(data=new\_df)

research\_impute\_numeric\_column(data2, col\_to\_fill, 5)



На основе графика для колонки Overall Rating можно сказать следующее:

* "мода" очень сильно завышает значимость плохих оценок, поэтому ее использовать не будем;
* "среднее" и константа 5 (а это в целом тоже средняя оценка в 10-бальной шкале) вырванивают датасет, добавляя в него средние значения. Из-за этого можно упустить рейс с очень плохим рейтингом или не знать о рейсе с превосходным рейтингом, поэтому эти статистики тоже не будем использовать для заполнения пропусков;
* "медиана" для оценок работает лучше всего, поэтому на данный момент это наилуйший кандидат для заполнения пропусков.

Далее еще посмотрим другие способы заполнения пропусков.

**Заполнение "хвостом распределения".** Этот метод бы подошел, если бы хотели выделить пропущенные значения из остальных значений, но у нас пропуски в наборе случайны. При это заполнение хвостом придаст значимость выбросам, чего нам не нужно.

**Добавления флага пропусков**. Этот метод добавляет еще один столбец, где пустые значения помечаются цифрой 1. Но нам все таки нужно именно заполнить пропуски.

**Заполнение значений для нескольких признаков**. У нас оценки независимы, поэтому этот метод тоже использовать не будем.

**Заполнение случайным значением признака**. Этот метод применяется для больших выборок, поэтому его также использовать не будем.

В итоге, аналогично анализируя числовые признаки, заполним все пропуски:

data2\_filled = data2.copy(deep=True)  
data2\_filled[number\_cols\_with\_na[0]] = impute\_column(data2, number\_cols\_with\_na[0], 'median')[0]  
data2\_filled[number\_cols\_with\_na[1]] = impute\_column(data2, number\_cols\_with\_na[1], 'mean')[0]  
data2\_filled[number\_cols\_with\_na[2]] = impute\_column(data2, number\_cols\_with\_na[2], 'mean')[0]  
data2\_filled[number\_cols\_with\_na[3]] = impute\_column(data2, number\_cols\_with\_na[3], 'median')[0]  
data2\_filled[number\_cols\_with\_na[4]] = impute\_column(data2, number\_cols\_with\_na[4], 'median')[0]  
data2\_filled[number\_cols\_with\_na[5]] = impute\_column(data2, number\_cols\_with\_na[5], 'median')[0]  
data2\_filled[number\_cols\_with\_na[6]] = impute\_column(data2, number\_cols\_with\_na[6], 'median')[0]  
data2\_filled[number\_cols\_with\_na[7]] = impute\_column(data2, number\_cols\_with\_na[7], 'median')[0]

Проверим, что пропуски были заполнены:

len([c for c in data2\_filled.columns if (data2\_filled[c].isnull().sum() > 0 and data2\_filled[c].dtype == 'float64')])

0

Число числовых колонок с пропусками = 0. Пропусков в числовых данных больше нет.

#### Категориальные признаки

**Заполнение самым частым значением или константой**

Константной будем заполнять в случаях, если пропуски имеют значение, а самым частым значением в ином случае.

* Для столбца 'Trip\_verified' na - имеет значение, а именно 'NotVerified'. То есть, если неизвестно, проверен ли отзыв - будем считать, что он не проверен. Заполним пропуски константой 'NotVerified';
* пропуски столбцов 'Aircraft', 'Origin', 'Destination', 'Date Flown', заполним значением 'NA' чтобы не завышать оценку самолету, пункту отправления/прибытия и дате;
* пропуски столбца 'Type Of Traveller' заполним наиболее частым значением, т.к. это не повлияет на набор, ведь уникальных типов путешественников всего 4.

data2\_filled['Trip\_verified'] = impute\_column(data2, 'Trip\_verified', 'constant', fill\_value\_param='NotVerified')[0]  
data2\_filled['Aircraft'] = impute\_column(data2, 'Aircraft', 'constant', fill\_value\_param='NA')[0]  
data2\_filled['Type Of Traveller'] = impute\_column(data2, 'Type Of Traveller', 'most\_frequent')[0]  
data2\_filled['Origin'] = impute\_column(data2, 'Origin', 'constant', fill\_value\_param='NA')[0]  
data2\_filled['Destination'] = impute\_column(data2, 'Destination', 'constant', fill\_value\_param='NA')[0]  
data2\_filled['Date Flown'] = impute\_column(data2, 'Date Flown', 'constant', fill\_value\_param='NA')[0]

Итог:

data2\_filled.isnull().sum().sum()

0

## Кодирование категориальных признаков

Кодирование категориальных признаков будем выполнять для датасета ryanair\_reviews

Категориальные признаки:

cat\_cols = [c for c in data2\_filled.columns if data2\_filled[c].dtype == 'O']  
cat\_cols

['Date Published',  
 'Passenger Country',  
 'Trip\_verified',  
 'Comment title',  
 'Comment',  
 'Aircraft',  
 'Type Of Traveller',  
 'Seat Type',  
 'Origin',  
 'Destination',  
 'Date Flown',  
 'Recommended']

data2\_enc = data2\_filled.copy(deep=True)

### Кодирование целочисленными значениями (label encoding)

Закодируем категории мест цифрами от 0 до 3. Таким образом, получаем, что, чем больше цифра, тем круче класс полета. Это можно использовать при каких-либо анализах

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()  
cat\_enc\_le = le.fit\_transform(data2\_filled['Seat Type'])

len(cat\_enc\_le)

2249

data2\_filled['Seat Type'].unique()

array(['Economy Class', 'Premium Economy', 'Business Class',  
 'First Class'], dtype=object)

Так можем посмотреть полученные цифры, которыми были закодированы категории

np.unique(cat\_enc\_le)

array([0, 1, 2, 3])

А так можно посмотреть исходные значения категорий

le.inverse\_transform([0, 1, 2, 3])

array(['Business Class', 'Economy Class', 'First Class',  
 'Premium Economy'], dtype=object)

Также далее будем использовать столбец Recommended в качестве целевого. Закодируем его 0лями и 1цами.

le2 = LabelEncoder()  
rec\_enc\_le = le2.fit\_transform(data2\_filled['Recommended'])  
np.unique(rec\_enc\_le)

array([0, 1])

le2.inverse\_transform([0, 1])

array(['no', 'yes'], dtype=object)

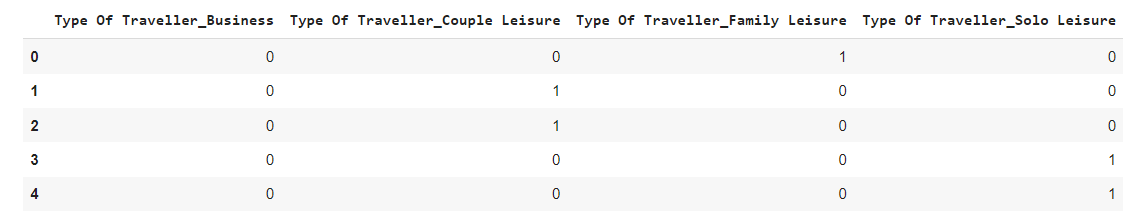
data2\_enc['Recommended'] = rec\_enc\_le

### Кодирование наборами бинарных значений (one-hot encoding)

One-hot encoding предполагает, что значение категории заменяется на отдельную колонку, которая содержит бинарные значения.

Приведем пример для столбца Type Of Traveller. В таком случае результат кодирования будет представлять собой набор строк, в которых стоит 1 для столбца, название которого и является значением исходного значения столбца 'Type Of Traveller' для этой строки.

pd.get\_dummies(data2\_filled[['Type Of Traveller']]).head()



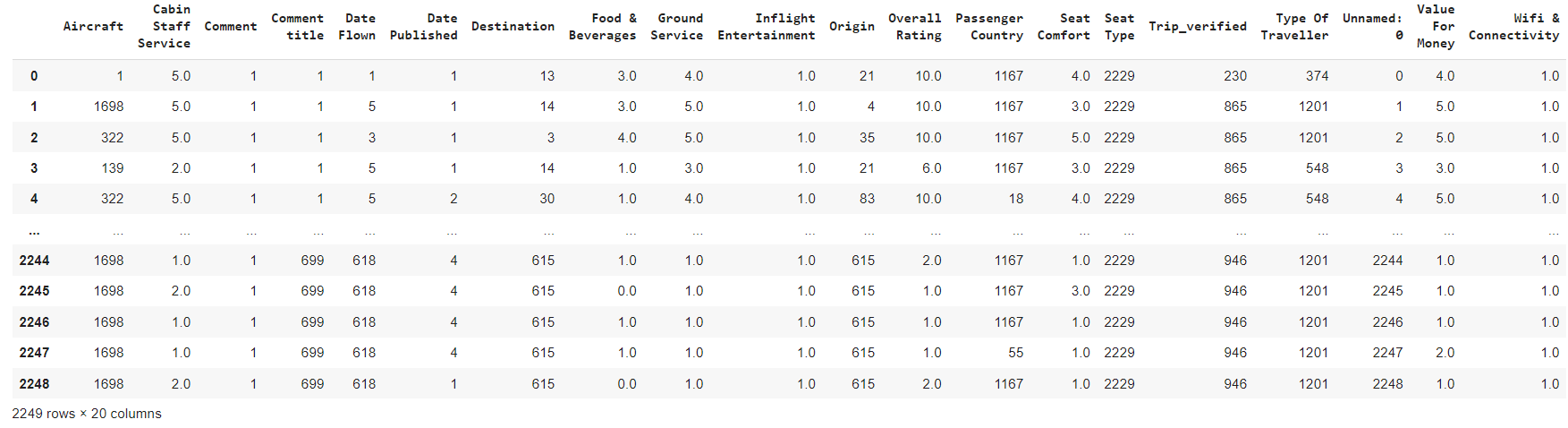
### Count (frequency) encoding

Count encoding предполагает что значение категории заменяется на количество раз, которое оно встречается в категории.

В случае frequency encoding вместо количества используется доля (процент) от количества записей.

Это очень полезные виды кодирования, которые помогут использовать значения частоты в разных вычислениях и анализах. Приведем пример для Recommended

ce\_CountEncoder1 = ce\_CountEncoder()  
data\_count\_enc = ce\_CountEncoder1.fit\_transform(data2\_enc[data2\_enc.columns.difference(['Recommended'])])  
data\_count\_enc

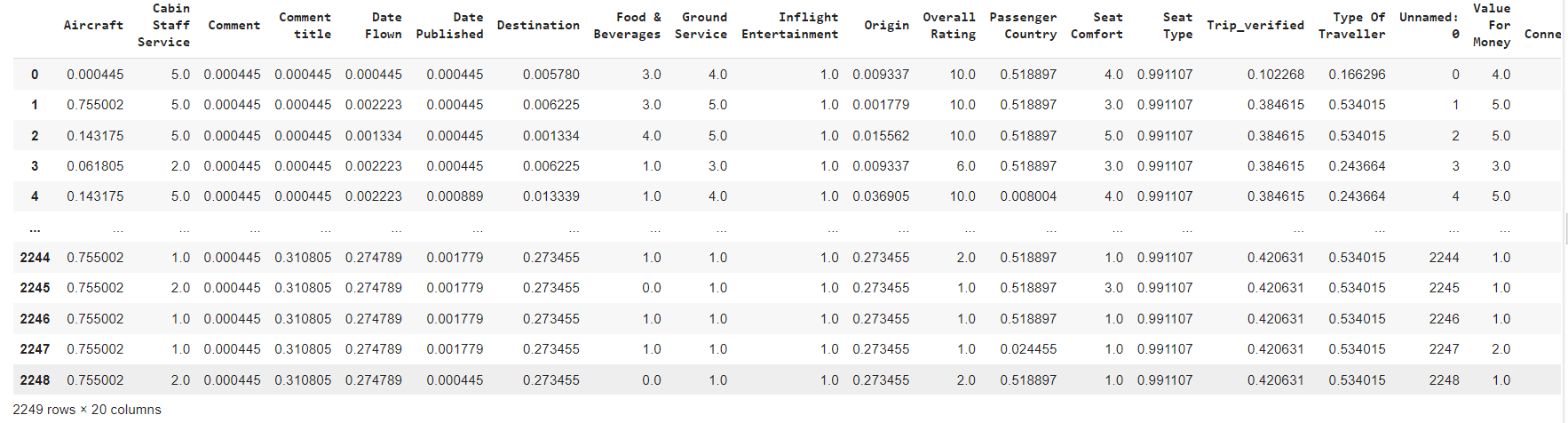


data\_count\_enc['Type Of Traveller'].unique()

array([ 374, 1201, 548, 126])

Таким образом получили количества встречающихся типов пассажиров

ce\_CountEncoder2 = ce\_CountEncoder(normalize=True)  
data\_freq\_enc = ce\_CountEncoder2.fit\_transform(data2\_enc[data2\_enc.columns.difference(['Recommended'])])  
data\_freq\_enc



data\_freq\_enc['Type Of Traveller'].unique()

array([0.16629613, 0.53401512, 0.24366385, 0.0560249 ])

Таким образом получили доли встречающихся типов пассажиров

### Target (Mean) encoding

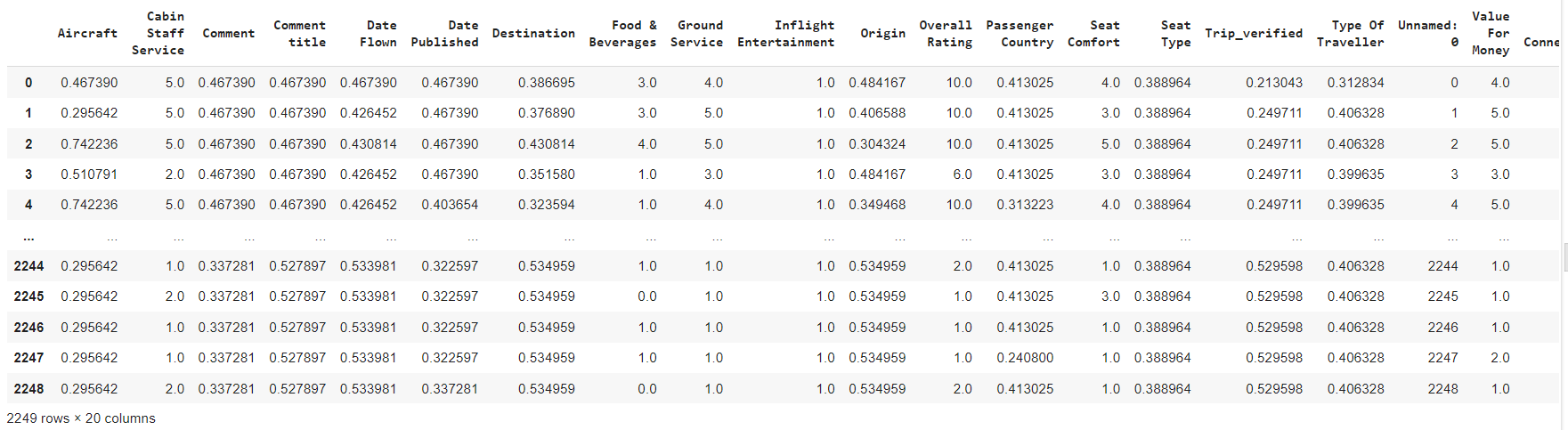
Target encoding предполагает что значение категории заменяется на ранг, соответствующий среднему значению целевого признака для этой категории.

В случае Mean encoding в кодируемый признак подставляется усредненное значение целевого признака.

Фактически, ранг зависит от вероятности появления целевого признака, при условии, что кодируемый признак принимает определенное значение.

Кодирование признака управляется значением целевого признака, возникает монотонная зависимость между кодируемым признаком и целевым признаком.

ce\_TargetEncoder1 = ce\_TargetEncoder()  
data\_mean\_enc = ce\_TargetEncoder1.fit\_transform(data2\_enc[data2\_enc.columns.difference(['Recommended'])], data2\_enc['Recommended'])



data2\_enc['Type Of Traveller'].unique()

array(['Family Leisure', 'Couple Leisure', 'Solo Leisure', 'Business'], dtype=object)

data\_mean\_enc['Type Of Traveller'].unique()

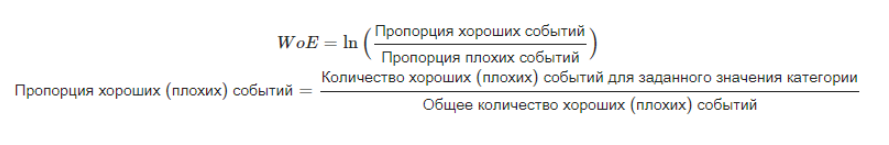
array([0.31283422, 0.40632806, 0.39963504, 0.38095255])

def check\_mean\_encoding(field):  
 for s in data2\_enc[field].unique():  
 data\_filter = data2\_enc[data2\_enc[field]==s]  
 if data\_filter.shape[0] > 0:  
 prob = sum(data\_filter['Recommended']) / data\_filter.shape[0]  
 print(s, '-' , prob)

check\_mean\_encoding('Type Of Traveller')

Family Leisure - 0.31283422459893045  
Couple Leisure - 0.4063280599500416  
Solo Leisure - 0.39963503649635035  
Business - 0.38095238095238093

### Weight of evidence (WoE) encoding

ce\_WOEEncoder1 = ce\_WOEEncoder()  
data\_woe\_enc = ce\_WOEEncoder1.fit\_transform(data2\_enc[data2\_enc.columns.difference(['Recommended'])], data2\_enc['Recommended'])

data2\_enc['Type Of Traveller'].unique()

array(['Family Leisure', 'Couple Leisure', 'Solo Leisure', 'Business'],  
 dtype=object)

data\_woe\_enc['Type Of Traveller'].unique()

array([-0.32624146, 0.07751303, 0.05056839, -0.02159405])

def check\_woe\_encoding(field):  
 data\_ones = data2\_enc[data2\_enc['Recommended'] == 1].shape[0]  
 data\_zeros = data2\_enc[data2\_enc['Recommended'] == 0].shape[0]  
  
 for s in data2\_enc[field].unique():  
 data\_filter = data2\_enc[data2\_enc[field]==s]  
 if data\_filter.shape[0] > 0:  
  
 filter\_data\_ones = data\_filter[data\_filter['Recommended'] == 1].shape[0]  
 filter\_data\_zeros = data\_filter[data\_filter['Recommended'] == 0].shape[0]  
  
 good = filter\_data\_ones / data\_ones  
 bad = filter\_data\_zeros / data\_zeros  
  
 woe = np.log(good/bad)  
 print(s, '-' , woe)

check\_woe\_encoding('Type Of Traveller')

Family Leisure - -0.33002907527778846  
Couple Leisure - 0.07770706025852656  
Solo Leisure - 0.04988705387080432  
Business - -0.02863474096202548

### Helmert encoding

Используется для упорядоченных категориальных переменных. Cравнивает каждый уровень категориальной переменной со средним значением последующих уровней. Для нашего набора этот метод не подойдет, т.к. у нас нет упорядоченных столбцов.

## Нормализация числовых признаков

Нормализация числового признака предполагает что на основе существующего признака мы создаем новый признак, который в идеале имеет нормальное распределение.

Нормализацию числовых признаков будем выполнять для датасета Air\_Traffic\_Passenger\_Statistics

Числовые признаки:

number\_cols = [c for c in data.columns if data[c].dtype == 'int64']  
number\_cols

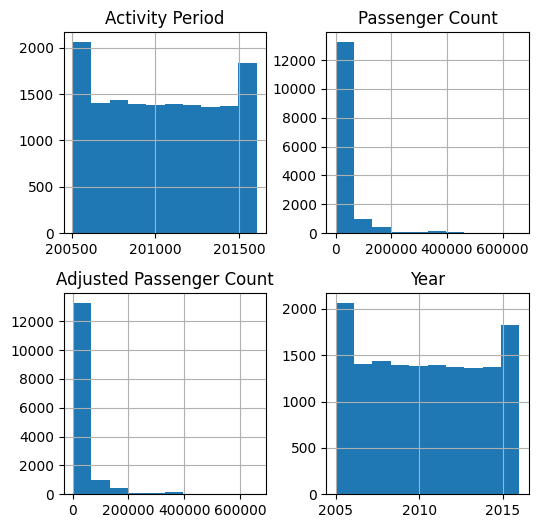
['Activity Period', 'Passenger Count', 'Adjusted Passenger Count', 'Year']

data\_number\_cols = data.copy(deep=True)[number\_cols]

Для начала проверим распределения наших числовых признаков

def diagnostic\_plots(df, variable):  
 plt.figure(figsize=(15,3))  
 # гистограмма  
 plt.subplot(1, 2, 1)  
 df[variable].hist(bins=30)  
 ## Q-Q plot  
 plt.subplot(1, 2, 2)  
 stats.probplot(df[variable], dist="norm", plot=plt)  
 plt.show()

data\_number\_cols.hist(figsize=(6,6))  
plt.show()

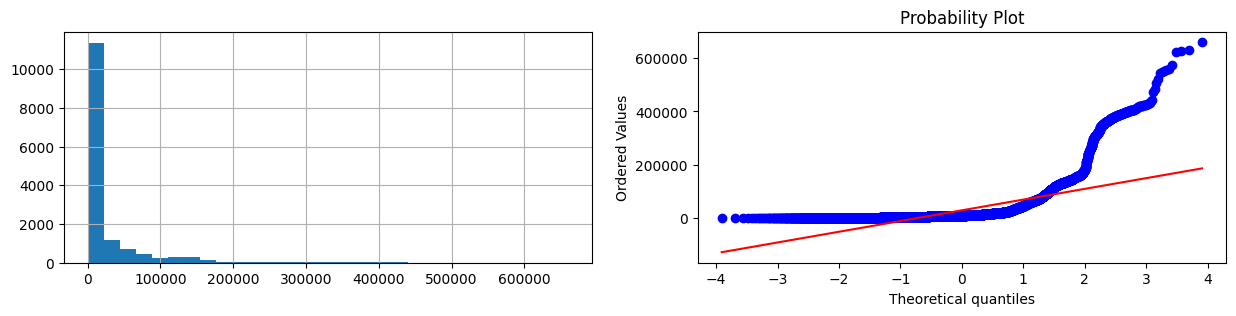


Далее будем рассматривать столбец Adjusted Passenger Count, как наиболее показательный и важный для анализа набора

col\_to\_norm = 'Adjusted Passenger Count'

Просмотрим гистограмму и Q-Q plot для одного из признаков

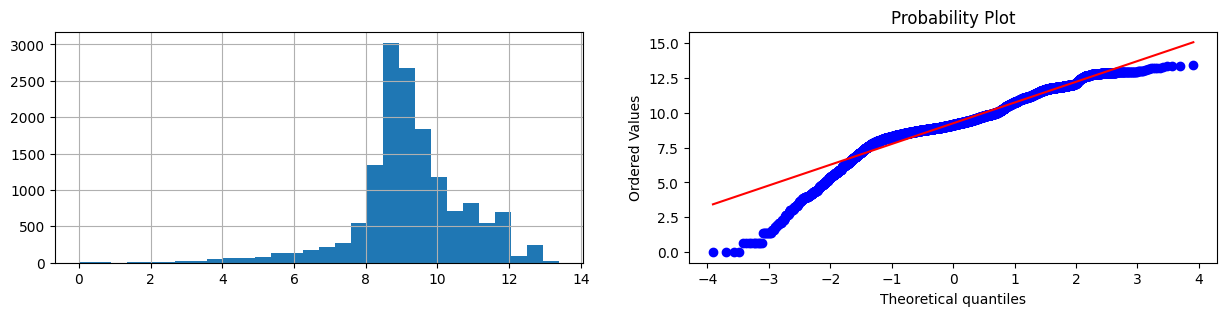
diagnostic\_plots(data\_number\_cols, col\_to\_norm)



Как видим, распределение отличается от нормального и есть несколько способов преобразовать его в нормальное

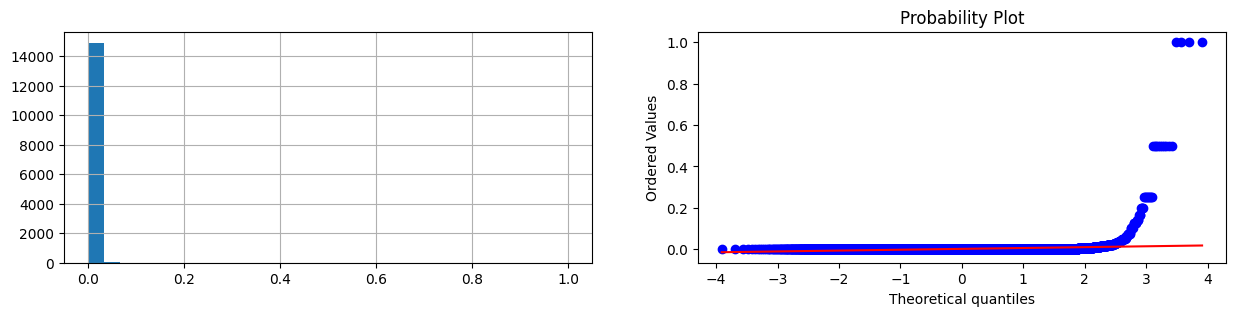
### Логарифмическое преобразование

data\_number\_cols['PassengerCount\_log'] = np.log(data\_number\_cols[col\_to\_norm] )  
diagnostic\_plots(data\_number\_cols, 'PassengerCount\_log')



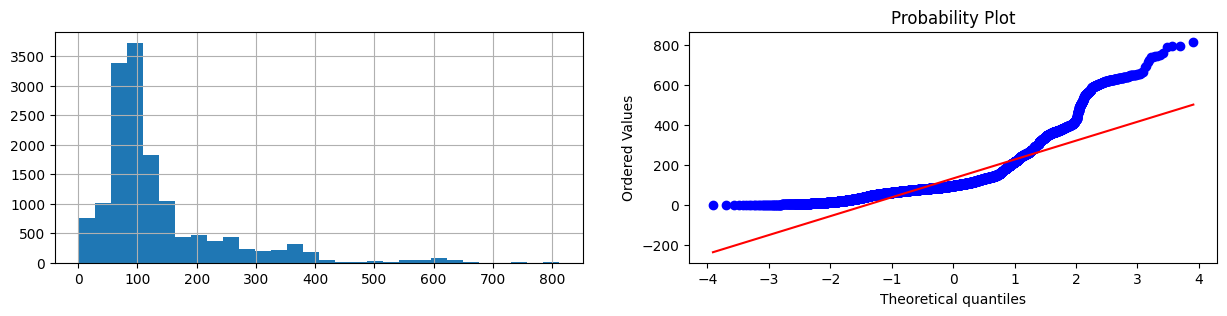
### Обратное преобразование

data\_number\_cols['PassengerCount\_obr'] = 1 / (data\_number\_cols[col\_to\_norm])  
diagnostic\_plots(data\_number\_cols, 'PassengerCount\_obr')



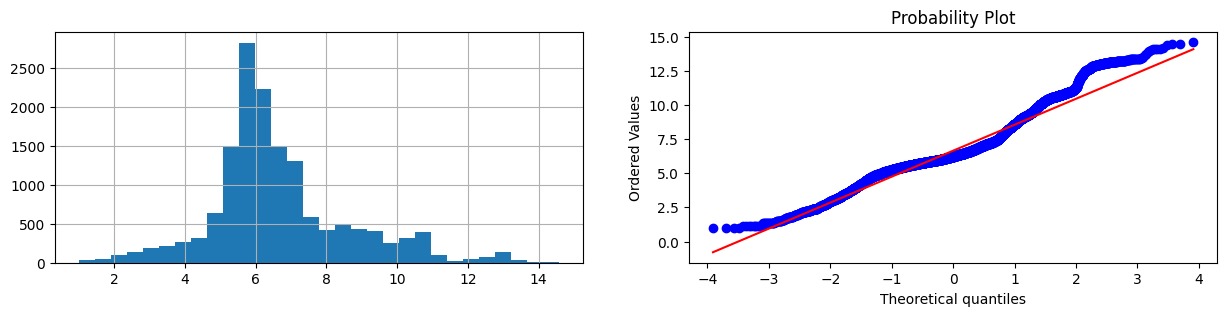
### Квадратный корень

data\_number\_cols['PassengerCount\_sqr'] = data\_number\_cols[col\_to\_norm]\*\*(1/2)  
diagnostic\_plots(data\_number\_cols, 'PassengerCount\_sqr')



### Возведение в степень

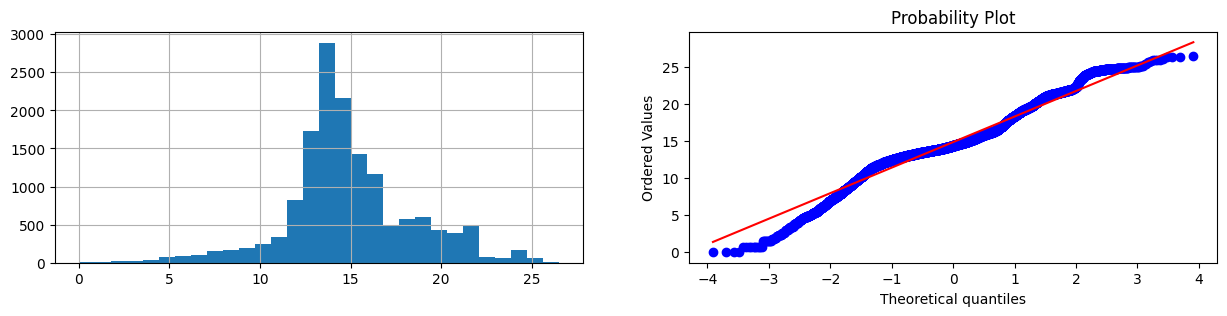
data\_number\_cols['PassengerCount\_exp'] = data\_number\_cols[col\_to\_norm]\*\*(1/5)  
diagnostic\_plots(data\_number\_cols, 'PassengerCount\_exp')



### Преобразование Бокса-Кокса

data\_number\_cols['PassengerCount\_bk'], param = stats.boxcox(data\_number\_cols[col\_to\_norm])  
print('Оптимальное значение λ = {}'.format(param))  
diagnostic\_plots(data\_number\_cols, 'PassengerCount\_bk')

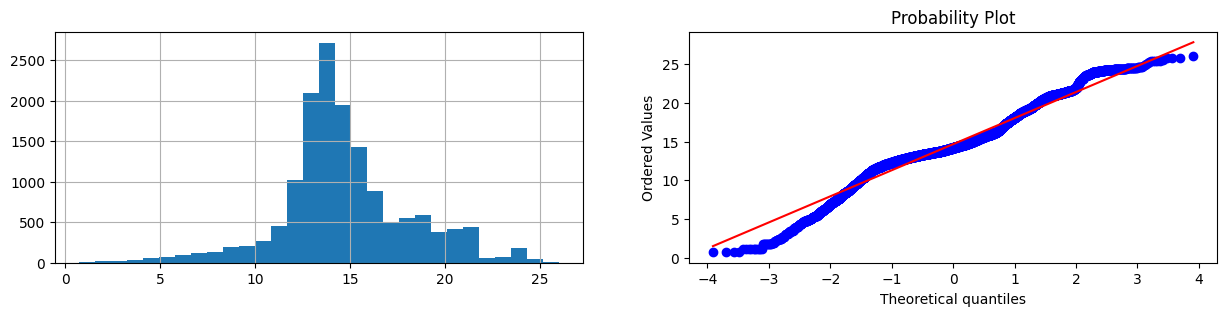
Оптимальное значение λ = 0.09242759544359987



### Преобразование Йео-Джонсона

data\_number\_cols[col\_to\_norm] = data\_number\_cols[col\_to\_norm].astype('float')  
data\_number\_cols['PassengerCount\_yedj'], param = stats.yeojohnson(data\_number\_cols[col\_to\_norm])  
print('Оптимальное значение λ = {}'.format(param))  
diagnostic\_plots(data\_number\_cols, 'PassengerCount\_yedj')

Оптимальное значение λ = 0.09012341596054435



### Вывод по нормализации

Лучше всего себя показали методы Бокса-Кокса и Йео-Джонсона, а также возведение в степень (но оптимальную степень еще нужно найти перебором)