Оглавление

[1. Введение 2](#_Toc167663213)

[2. Обработка данных 2](#_Toc167663214)

[2.1. Структура данных для анализа вакансий и кандидатов 2](#_Toc167663215)

[2.2. Преобразование JSON вакансий и резюме в DataFrame 6](#_Toc167663216)

[2.3. Предобработка данных 8](#_Toc167663217)

[2.4. Векторизация данных 21](#_Toc167663218)

[3. Заключение 22](#_Toc167663219)

# Введение

Анализ вакансий и резюме является ключевым компонентом в управлении человеческими ресурсами и процессе найма. Для эффективного анализа и обработки этих данных часто используется структурированный формат данных, такой как JSON. Файл data.json содержит информацию о вакансиях и откликнувшихся на них кандидатах, организованную в виде объектов. Каждый объект описывает отдельную вакансию и сопутствующую информацию о кандидатах.

В этом документе будет подробно рассмотрена структура данных, представленных в файле data.json, включая основные элементы и их характеристики. Также будет описан процесс преобразования этих данных в формат датафрейма с использованием библиотеки pandas и последующей предобработки данных для дальнейшего анализа и моделирования.

# Обработка данных

## Структура данных для анализа вакансий и кандидатов

Файл data.json содержит информацию о вакансиях и откликнувшихся на них кандидатах. Структура файла организована в виде объектов, каждый из которых описывает отдельную вакансию и сопутствующую информацию о кандидатах. Далее будет разобрана структура данных, представленных в файле.

**Основные элементы структуры JSON**

1. Вакансия (vacancy):

* uuid: Уникальный идентификатор вакансии.
* name: Название вакансии.
* keywords: Ключевые слова (могут отсутствовать).
* description: Описание вакансии, включая задачи, требования и условия работы.
* comment: Комментарий к вакансии (например, информация о зарплате).

2. Неуспешные резюме (failed\_resumes):

Список резюме, которые не прошли отбор по данной вакансии. Каждый элемент содержит:

* uuid: Уникальный идентификатор резюме.
* first\_name: Имя кандидата.
* last\_name: Фамилия кандидата.
* birth\_date: Дата рождения кандидата.
* country: Страна проживания.
* city: Город проживания.
* about: Дополнительная информация о кандидате (биографическая справка).
* key\_skills: Ключевые навыки кандидата.
* experienceItem: Список опыта работы, где каждый элемент включает: starts (дата начала работы) и ends: (дата окончания работы)
* employer: Название компании.
* city: Город, в котором работал кандидат.
* position: Должность кандидата.
* description: Описание обязанностей и достижений на данной должности.
* educationItem: Список образования, где каждый элемент включает:
* year: Год окончания учебного заведения.
* organization: Название учебного заведения.
* faculty: Факультет.
* specialty: Специальность.
* result: Результаты обучения (могут отсутствовать).
* education\_type: Тип образования (основное, повышение квалификации и т.д.).
* education\_level: Уровень образования (высшее, среднее специальное и т.д.).

3. Успешные резюме (confirmed\_resumes):

Список резюме, которые успешно прошли отбор по данной вакансии. Структура аналогична элементам из списка неуспешных резюме.

**Пример данных из JSON**

Пример описания вакансии:

|  |
| --- |
| {  "vacancy": {  "uuid": "779f3a59-206a-3241-adc4-d7db504f960b",  "name": "Java разработчик команда Инвестиции",  "keywords": null,  "description": "Описание Мы расширяем команды и ищем разработчиков...",  "comment": "450 на руки"  },  "failed\_resumes": [  {  "uuid": "74392e00-ecfb-335b-9fc1-c2652dca06e5",  "first\_name": "Данила",  "last\_name": "Прохоров",  "birth\_date": "1979-01-01",  "country": "Россия",  "city": "Санкт-Петербург",  "about": null,  "key\_skills": "Java, Spring Boot, Java EE, SQL...",  "experienceItem": [  {  "starts": "2022-05-01",  "ends": "2023-10-23",  "employer": "МФО ХмельИнфоОрион",  "city": "Можайск",  "position": "Ведущий инженер-программист",  "description": "Интеграционные адаптеры для передачи заявок пользователей..."  },  // другие элементы опыта работы  ],  "educationItem": [  {  "year": 1999,  "organization": "СибГАУ",  "faculty": "Информатики и систем управления",  "specialty": "",  "result": "",  "education\_type": "Основное",  "education\_level": "Высшее"  }  ]  },  // другие неуспешные резюме  ],  "confirmed\_resumes": [  {  "uuid": "8c8cf797-2c6b-3f4b-b28b-20d57bd88b82",  "first\_name": "Марк",  "last\_name": "Силин",  "birth\_date": "1987-01-01",  "country": "Россия",  "city": "Великий Новгород",  "about": null,  "key\_skills": "Java, Java Spring Framework, Hibernate...",  "experienceItem": [  {  "starts": "2022-02-01",  "ends": null,  "employer": "ОАО ЭлектроАвтоТяж",  "city": "Коломна",  "position": "java-разработчик",  "description": "Проекты: Внутренний продукт по заказу инструментов разработки для сотрудников..."  },  // другие элементы опыта работы  ],  "educationItem": [  {  "year": null,  "organization": "НовГУ",  "faculty": "Архитектуры, искусств и строительства",  "specialty": "",  "result": "",  "education\_type": "Основное",  "education\_level": "Высшее"  }  ]  },  // другие успешные резюме  ]  } |

Листинг 1

**Основные характеристики данных**

* Идентификаторы: Все элементы имеют уникальные идентификаторы (UUID).
* Детализированное описание: Включает подробное описание вакансий и опыта кандидатов.
* Формат данных: Используется JSON формат, обеспечивающий структурированное и легко читаемое представление данных.
* Разделение резюме: Резюме разделены на успешные и неуспешные, что позволяет легко фильтровать и анализировать кандидатов по результатам отбора.

Этот файл предоставляет обширную информацию для анализа вакансий и откликнувшихся кандидатов, что может быть полезно для HR-аналитики и управления процессом найма.

## Преобразование JSON вакансий и резюме в DataFrame

Процесс преобразования JSON файла, содержащего информацию о вакансиях и резюме, в формат датафрейма с использованием библиотеки pandas, выполняется следующим образом:

1. Чтение JSON файла:

|  |
| --- |
| # Читаем json файл с резюме и вакансиями  with open('data.json', 'r', encoding='utf-8') as f:  data = json.load(f) |

Листинг 2

Сначала файл data.json открывается для чтения в текстовом режиме с использованием кодировки utf-8.

Содержимое файла загружается в переменную data с помощью функции json.load, которая преобразует JSON строку в соответствующую структуру данных Python (в данном случае это будет список словарей).

2. Извлечение отклонённых резюме:

|  |
| --- |
| # Извлекаем отклонённые резюме  df\_f = pd.json\_normalize(data, record\_path=['failed\_resumes'], meta=[['vacancy', 'uuid'], ['vacancy', 'name'], ['vacancy', 'keywords'], ['vacancy', 'description'], ['vacancy', 'comment']]) |

Листинг 3

Используется функция pd.json\_normalize для нормализации вложенных данных JSON.

record\_path=['failed\_resumes'] указывает, что мы хотим извлечь данные из ключа failed\_resumes.

meta параметр указывает, какие дополнительные данные из объекта vacancy должны быть добавлены к каждой строке результирующего датафрейма. Эти данные включают:

* vacancy.uuid - UUID вакансии.
* vacancy.name - Название вакансии.
* vacancy.keywords - Ключевые слова вакансии.
* vacancy.description - Описание вакансии.
* vacancy.comment - Комментарий к вакансии.

3. Извлечение принятых резюме:

|  |
| --- |
| # Извлекаем принятые резюме  df\_c = pd.json\_normalize(data, record\_path=['confirmed\_resumes'], meta=[['vacancy', 'uuid'], ['vacancy', 'name'], ['vacancy', 'keywords'], ['vacancy', 'description'], ['vacancy', 'comment']]) |

Листинг 4

Аналогично предыдущему шагу, извлекаются данные из ключа confirmed\_resumes с добавлением тех же метаданных из объекта vacancy.

4. Добавление кодов статусов

|  |
| --- |
| # Добавляем коды статусов  df\_f['status'] = 0  df\_c['status'] = 1 |

Листинг 5

Создаются новые столбцы status в датафреймах df\_f и df\_c для указания статуса резюме:

* 0 для отклонённых резюме.
* 1 для принятых резюме.

5. Сборка единого датафрейма:

|  |
| --- |
| # Собираем в единый датасет  frames = [df\_f, df\_c]  df = pd.concat(frames) |

Листинг 6

Создаётся список frames, содержащий оба датафрейма df\_f и df\_c.

Функция pd.concat объединяет эти датафреймы в один общий датафрейм df.

**Итоговый DataFrame**

В результате выполнения вышеописанного кода, создаётся объединённый датафрейм df, который включает данные обо всех резюме (как отклонённых, так и принятых) с добавленными метаданными о вакансиях и статусом каждого резюме.

Такой датафрейм предоставляет удобный способ для анализа и работы с данными о вакансиях и кандидатах.

## Предобработка данных

После загрузки данных и преобразования их в формат датафрейма, необходимо провести этап предобработки данных. В данном разделе будут подробно рассмотрены все шаги, выполненные для подготовки данных к дальнейшему анализу и моделированию.

**Шаги предобработки данных**

1. Заполнение пропущенных значений

Пропущенные значения в датафрейме заполняются нулями:

|  |
| --- |
| df = df.fillna(0) |

Листинг 7

2. Удаление ненужных столбцов

Удаляются столбцы, которые не несут значимой информации для анализа и машинного обучения:

|  |
| --- |
| df = df.drop(['uuid', 'first\_name', 'last\_name', 'country'], axis=1) |

Листинг 8

Данный шаг помогает уменьшить размер данных, избежать избыточности и повысить эффективность анализа. В нашем случае мы удаляем следующие столбцы: uuid, first\_name, last\_name и country.

Столбец uuid содержит уникальный идентификатор для каждого резюме. Этот идентификатор используется исключительно для идентификации записей и не несет никакой значимой информации для анализа или моделирования. Поскольку uuid не влияет на предсказательную мощность моделей и не содержит полезной информации о характеристиках кандидатов или вакансий, то данный столбец подлежит удалению.

Столбец first\_name содержит имя кандидата. Имена кандидатов, как правило, не несут значимой информации для анализа и могут рассматриваться как личные данные, которые необходимо исключить для обеспечения конфиденциальности. В контексте машинного обучения имена не оказывают влияния на исход моделирования, поэтому данный столбец также удаляется.

Столбец last\_name содержит фамилию кандидата. Аналогично столбцу first\_name, фамилии не несут аналитической ценности и могут рассматриваться как конфиденциальные данные. В целях повышения конфиденциальности и исключения избыточной информации данный столбец удаляется.

Столбец country содержит информацию о стране проживания кандидата. В данном контексте, информация о стране является избыточной, особенно учитывая тот факт, что все данные относятся к одной стране. В нашем случае, этот столбец удаляется, чтобы упростить данные и сосредоточиться на более релевантных характеристиках кандидатов и вакансий.

3. Расчет возраста кандидатов

Создается функция calculate\_age, которая вычисляет возраст кандидата на основе даты рождения.

|  |
| --- |
| def calculate\_age(birthdate):  if birthdate == 0:  return 25  else:  birthdate = datetime.datetime.strptime(birthdate, "%Y-%m-%d")  today = datetime.datetime.today()  return today.year - birthdate.year - ((today.month, today.day) < (birthdate.month, birthdate.day))  df['age'] = df['birth\_date'].apply(calculate\_age)  df = df.drop(['birth\_date'], axis=1) |

Листинг 9

Функция calculate\_age принимает на вход строку с датой рождения кандидата. Она проверяет, является ли дата рождения недоступной (обозначена как 0). Если дата отсутствует, функция возвращает значение 25 лет. Если дата рождения указана, функция вычисляет возраст кандидата на основе текущей даты.

Возраст вычисляется путем вычитания года рождения из текущего года. Дополнительно учитывается, прошел ли день рождения в текущем году. Если текущий месяц и день меньше месяца и дня рождения, то возраст уменьшается на 1.

После того, как функция calculate\_age определена, она применяется к каждому элементу столбца birth\_date в датафрейме df. Для этого используется метод apply.

После вычисления возраста кандидатов, столбец birth\_date больше не нужен, поэтому его можно удалить.

4. Объединение языковых данных

Создается функция combine\_columns, которая объединяет данные о языках из двух разных столбцов:

|  |
| --- |
| def combine\_columns(row):  if row['languageItem'] == 0 and row['languageItems'] == 0:  return 0  elif row['languageItem'] == 0 and row['languageItems'] != 0:  return row['languageItems']  elif row['languageItem'] != 0 and row['languageItems'] == 0:  return row['languageItem']  else:  return str(row['languageItem']) + str(row['languageItems'])  df['Languages'] = df.apply(combine\_columns, axis=1)  df = df.drop(['languageItems', 'languageItem'], axis=1) |

Листинг 10

В процессе предобработки данных часто возникает необходимость объединить информацию, разбросанную по нескольким столбцам, в один. В данном случае, данные о языках кандидатов хранятся в двух столбцах: languageItem и languageItems. Для их объединения создается функция combine\_columns, которая аккумулирует информацию из этих двух столбцов в новый столбец Languages.

Подробное описание этапов:

Функция combine\_columns принимает на вход одну строку датафрейма и объединяет данные о языках из двух столбцов: languageItem и languageItems. Логика функции заключается в следующем:

Если оба столбца равны 0, функция возвращает 0. Это означает, что у кандидата нет данных о языках.

Если languageItem равен 0, а languageItems содержит данные, функция возвращает значение из languageItems.

Если languageItem содержит данные, а languageItems равен 0, функция возвращает значение из languageItem.

Если оба столбца содержат данные, функция объединяет их, преобразуя в строку и конкатенируя.

5. Расчет опыта работы

Создается функция calculate\_experience, которая вычисляет общий стаж работы кандидата на основе записей о предыдущих местах работы:

|  |
| --- |
| def calculate\_experience(job\_records):  if job\_records == 0:  return 0  else:  total\_experience = 0  for job in job\_records:  start\_date = datetime.datetime.strptime(job['starts'], '%Y-%m-%d')  if job['ends'] is None:  end\_date = datetime.datetime.now() # Current date if 'ends' is None  else:  end\_date = datetime.datetime.strptime(job['ends'], '%Y-%m-%d')  duration = (end\_date - start\_date).days / 365.25 # Convert days to years  total\_experience += duration  return total\_experience  df['years\_of\_experience'] = df['experienceItem'].apply(calculate\_experience) |

Листинг 11

Общий стаж работы кандидата является одним из ключевых факторов при анализе и принятии решений в процессе найма. Для вычисления общего стажа работы кандидатов создается функция calculate\_experience, которая на основе записей о предыдущих местах работы подсчитывает количество лет, которые кандидат провел на различных должностях.

Функция calculate\_experience принимает на вход список записей о предыдущих местах работы кандидата (job\_records). Она проверяет, является ли список пустым (значение 0). Если список пустой, функция возвращает 0, иначе вычисляет общий стаж работы.

6. Обработка данных об образовании

Для предобработки данных об образовании создаются несколько функций, каждая из которых извлекает определенные аспекты информации об учебных заведениях, годах окончания, уровнях образования и специальностях. Эти функции применяются к столбцу educationItem в датафрейме df.

|  |
| --- |
| def university\_str(a):  if a == 0:  return 0  else:  return str(a)  def university(a):  if a == 0:  return 0  else:  a = a.split(",")  if len(a) < 1:  return 0  else:  a = a[1]  a = a.replace("organization","").replace(':',"").replace("'","")  return a  def university\_year(a):  if a == 0:  return 0  else:  a = a.split(",")  if len(a) < 1:  return 0  else:  a = a[0]  a = a.replace("year","").replace(':',"").replace("'","").replace("[","").replace("{","")  return a  def university\_level(a):  if "Высшее" in str(a):  return 1  else:  return 0  def extract\_specialty(data\_string):  data\_string = str(data\_string)  pattern = re.compile(r"'specialty': '(.\*?)'")  match = pattern.search(data\_string)  return match.group(1) if match else 0  df["university"] = df["educationItem"].apply(university\_str).apply(university)  df["university\_year"] = df["educationItem"].apply(university\_str).apply(university\_year)  df["university\_level"] = df["educationItem"].apply(university\_str).apply(university\_level)  df["specialty"] = df["educationItem"].apply(extract\_specialty) |

Листинг 12

**Функция university\_str:**

Эта функция принимает значение и проверяет, является ли оно равным 0. Если значение равно 0, функция возвращает 0. В противном случае она преобразует значение в строку. Это необходимо, чтобы унифицировать формат данных и обеспечить их правильное дальнейшее разбиение и обработку.

Функция university:

Эта функция извлекает название учебного заведения из строки. Она проверяет, является ли значение равным 0. Если значение равно 0, функция возвращает 0. В противном случае строка разбивается на части с использованием запятой в качестве разделителя. Если длина полученного списка меньше 1, функция возвращает 0. В противном случае она извлекает второй элемент списка (который соответствует названию учебного заведения), удаляет ненужные части строки и возвращает результат.

Функция university\_year:

Эта функция извлекает год окончания учебного заведения. Она проверяет, является ли значение равным 0. Если значение равно 0, функция возвращает 0. В противном случае строка разбивается на части с использованием запятой. Если длина списка меньше 1, функция возвращает 0. В противном случае извлекается первый элемент списка (который соответствует году окончания), удаляются ненужные части строки, и результат возвращается.

Функция university\_level:

Эта функция определяет уровень образования на основе строки. Если в строке содержится подстрока "Высшее", функция возвращает 1. В противном случае функция возвращает 0. Это позволяет закодировать уровень образования в бинарный формат.

Функция extract\_specialty:

Эта функция использует регулярное выражение для извлечения специальности из строки. Она преобразует входное значение в строку, определяет шаблон для поиска специальности и ищет его в строке. Если специальность найдена, функция возвращает ее значение. В противном случае возвращается 0.

Применение функций к датафрейму:

После определения всех необходимых функций они применяются к столбцу educationItem в датафрейме df. Каждая функция сначала применяется для преобразования данных в строку с помощью university\_str, а затем — для извлечения соответствующей информации.

7. Создание сводного описания опыта работы

Создается функция extract\_summary, которая создает сводное описание опыта работы кандидата:

|  |
| --- |
| def extract\_summary(experiences):  if not isinstance(experiences, list):  return "Invalid data format"  summaries = []  for experience in experiences:  start = experience.get('starts', 'Unknown start date')  end = experience.get('ends', 'Present') or 'Present'  employer = experience.get('employer', 'Unknown employer')  city = experience.get('city', 'Unknown city')  position = experience.get('position', 'Unknown position')  description = experience.get('description', 'No description provided').split('.')[0] + '...' if experience.get('description') else 'No description provided.'  summary = f"{position} at {employer}, {city} ({start} - {end}): {description}"  summaries.append(summary)  return ' | '.join(summaries) if summaries else "No work experience provided"  df['work\_summary'] = df['experienceItem'].apply(extract\_summary) |

Листинг 13

Для создания сводного описания опыта работы кандидатов создается функция extract\_summary. Эта функция принимает на вход список записей о предыдущих местах работы кандидата и возвращает строку, содержащую краткие описания каждого опыта. Если входное значение не является списком, функция возвращает сообщение о неверном формате данных.

Описание работы функции:

1. Проверка формата входных данных

Функция сначала проверяет, является ли входное значение списком. Если это не так, возвращается сообщение о неверном формате данных. Это необходимо для обеспечения корректной обработки данных и предотвращения ошибок.

1. Инициализация списка для хранения сводных описаний

Если входное значение является списком, инициализируется пустой список summaries, который будет использоваться для хранения сводных описаний каждого опыта работы.

1. Цикл по каждой записи о работе

Функция проходит по каждой записи в списке experiences и извлекает необходимые данные:

Дата начала работы: извлекается из ключа starts. Если ключ отсутствует, используется значение 'Unknown start date'.

Дата окончания работы: извлекается из ключа ends. Если ключ отсутствует или равен None, используется значение 'Present'.

Название компании: извлекается из ключа employer. Если ключ отсутствует, используется значение 'Unknown employer'.

Город работы: извлекается из ключа city. Если ключ отсутствует, используется значение 'Unknown city'.

Должность: извлекается из ключа position. Если ключ отсутствует, используется значение 'Unknown position'.

Описание обязанностей: извлекается из ключа description. Если ключ отсутствует, используется значение 'No description provided'. Описание обрезается до первого предложения и добавляется многоточие для краткости.

1. Создание сводного описания для каждой записи

Для каждой записи создается строка с кратким описанием опыта работы в формате: "<Должность> at <Компания>, <Город> (<Дата начала> - <Дата окончания>): <Описание>". Эта строка добавляется в список summaries.

1. Объединение всех сводных описаний

После обработки всех записей, все строки из списка summaries объединяются в одну строку с разделителем «|». Если список summaries пуст, возвращается сообщение "No work experience provided".

1. Применение функции к датафрейму

После определения функции extract\_summary, она применяется к столбцу experienceItem в датафрейме df. Результат сохраняется в новый столбец work\_summary.

10. Удаление обработанных столбцов

После извлечения и преобразования всех необходимых данных, удаляются столбцы с исходными записями:

|  |
| --- |
| df = df.drop(['experienceItem', 'educationItem', 'vacancy.comment'], axis=1) |

Листинг 14

11. Преобразование категориальных признаков

В процессе предобработки данных для машинного обучения важно преобразовать категориальные признаки в числовые, поскольку большинство алгоритмов машинного обучения работают только с числовыми данными. Для этого используется метод LabelEncoder из библиотеки sklearn. Он заменяет каждое уникальное значение в столбце на числовой код.

**Обоснование выбора метода LabelEncoder**

Метод LabelEncoder выбран по следующим причинам:

1. Простота и эффективность: LabelEncoder предоставляет простой и быстрый способ преобразования категориальных данных в числовые коды. Он присваивает каждому уникальному значению в столбце уникальный числовой код.

2. Сохранение порядка уникальных значений: LabelEncoder сохраняет порядок уникальных значений в исходном столбце, что может быть полезно для некоторых моделей и алгоритмов.

3. Поддержка широкого спектра данных: LabelEncoder может обрабатывать строки и числовые значения, что делает его универсальным инструментом для преобразования различных типов категориальных данных.

**Подробное описание этапов**

1. Создание словаря для хранения LabelEncoder

Сначала создается пустой словарь label\_encoders, который будет использоваться для хранения объектов LabelEncoder для каждого столбца. Это позволяет сохранить доступ к каждому LabelEncoder для обратного преобразования или повторного использования.

2. Определение категориальных столбцов

Задается список категориальных столбцов, которые необходимо преобразовать. В данном случае это столбцы: city, vacancy.uuid, vacancy.name, Languages, university, specialty.

3. Преобразование категориальных данных

Проход по каждому столбцу из списка categorical\_columns и выполнение следующих шагов:

**Создание экземпляра LabelEncoder.**

Преобразование значений столбца в строковый формат с помощью метода astype(str), чтобы обеспечить корректную работу LabelEncoder.

Применение метода fit\_transform для преобразования значений столбца в числовые коды.

Сохранение объекта LabelEncoder в словаре label\_encoders для последующего использования.

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder    for col in categorical\_columns:  le = LabelEncoder()  df[col] = le.fit\_transform(df[col].astype(str))  label\_encoders[col] = le |

Листинг 15

Каждый шаг преобразования категориальных данных выполняется внутри цикла for, который проходит по всем столбцам, указанным в categorical\_columns.

**Преобразование значений столбца в строковый формат**

Для обеспечения корректной работы LabelEncoder все значения столбца преобразуются в строковый формат:

|  |
| --- |
| df[col] = df[col].astype(str) |

Листинг 16

**Применение метода fit\_transform**

Метод fit\_transform обучает LabelEncoder на данных столбца и одновременно преобразует их в числовые коды:

|  |
| --- |
| df[col] = le.fit\_transform(df[col].astype(str)) |

Листинг 17

**Полный код**

Весь процесс преобразования категориальных признаков выглядит следующим образом:

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  label\_encoders = {}  categorical\_columns = ['city', 'vacancy.uuid', 'vacancy.name', 'Languages', 'university', 'specialty']  for col in categorical\_columns:  le = LabelEncoder()  df[col] = le.fit\_transform(df[col].astype(str))  label\_encoders[col] = le |

Листинг 18

После выполнения этих шагов все категориальные столбцы в датафрейме df будут преобразованы в числовые коды. Это позволяет использовать эти данные для дальнейшего анализа и построения моделей машинного обучения. Кроме того, словарь label\_encoders сохраняет объекты LabelEncoder для каждого столбца, что позволяет легко выполнять обратное преобразование кодов в исходные значения при необходимости.

## Векторизация данных

После предварительной обработки данных в датафрейме остаются четыре текстовых столбца: vacancy.description, about, key\_skills, и work\_summary. Векторизация этих столбцов необходима для того, чтобы преобразовать текстовые данные в числовой формат, который может быть использован в алгоритмах машинного обучения.

**Почему векторизация не включена в этап препроцессинга**

Векторизация текстовых столбцов с использованием дообученной версии модели BERT (Bidirectional Encoder Representations from Transformers) не включена в этап препроцессинга по следующим причинам:

**Обучение модели**

Векторизация с использованием BERT требует предварительного обучения модели на текстовых данных. Это означает, что модель должна быть дообучена на данных, чтобы обеспечить высокое качество векторных представлений. Процесс дообучения модели BERT считается частью обучения, а не предобработки данных.

**Требования к вычислительным ресурсам**

Обучение и использование моделей глубокого обучения, таких как BERT, требует значительных вычислительных ресурсов, включая использование графических процессоров (GPU). Это делает этот процесс более ресурсоемким и подходящим для этапа обучения моделей, а не для предобработки данных.

**Сложность и специфика обработки**

Векторизация текста с помощью моделей BERT включает сложные процессы, такие как токенизация текста, добавление специальных токенов (например, [CLS], [SEP]), и использование специфических архитектур трансформеров для получения контекстуально зависимых представлений слов. Эти процессы выходят за рамки стандартных методов предобработки данных и требуют специализированного подхода.

**Основные этапы векторизации с использованием BERT**

Хотя векторизация с использованием BERT не является частью предобработки, важно уточнить основные этапы этого процесса:

* Загрузка предобученной модели BERT:

Модель BERT, предварительно обученная на больших корпусах текста (например, Wikipedia, BookCorpus), загружается из библиотеки transformers.

* Токенизация текста:

Использование токенизатора BERT для разбиения текста на токены, добавления специальных токенов ([CLS], [SEP]), и создания масок внимания.

* Применение модели BERT:

Пропускание токенизированного текста через модель BERT для получения векторных представлений (эмбеддингов) для каждого токена. Обычно используется эмбеддинг [CLS]-токена для представления всего текста.

* Сохранение эмбеддингов:

Сохранение полученных эмбеддингов для использования в дальнейших моделях машинного обучения.

# Заключение

В ходе работы были проведены все необходимые этапы предобработки данных для анализа вакансий и резюме. Начав с загрузки и преобразования данных из формата JSON в DataFrame, мы выполнили очистку данных, обработку пропущенных значений, удаление ненужных столбцов, расчет возраста кандидатов, объединение данных о языках, расчет общего стажа работы и извлечение данных об образовании.

Особое внимание уделено преобразованию категориальных признаков в числовые коды с использованием LabelEncoder, что подготовило данные для последующего анализа и моделирования. В итоге остались текстовые столбцы (vacancy.description, about, key\_skills, work\_summary), которые будут векторизованы с помощью дообученной версии модели BERT. Этот шаг выполняется отдельно, так как векторизация с использованием BERT включает обучение модели и требует значительных вычислительных ресурсов.

Подготовленные данные теперь готовы для углубленного анализа и построения моделей машинного обучения, что позволит эффективно анализировать и предсказывать результаты найма, а также улучшить процесс подбора персонала. Весь процесс предобработки данных обеспечивает высокое качество данных и готовность к дальнейшему использованию в аналитических и предсказательных задачах.