Вопрос № 1. Рассмотрите основные особенности обучения моделей машинного обучения на размеченных данных.

1. Размеченные данные: Для обучения модели требуются размеченные данные, то есть данные, для которых известны правильные ответы или целевые значения. Эти данные должны быть представлены в виде признаков (входных переменных) и соответствующих им меток (выходных переменных). Разметка может быть выполнена экспертами вручную или с использованием автоматических методов.
2. Обучение на основе примеров: Модель машинного обучения обучается на основе образцов из размеченных данных. Она ищет закономерности и шаблоны в этих данных, чтобы предсказывать правильные ответы для новых, ранее не виденных данных.
3. Выбор модели: Существует множество алгоритмов машинного обучения, и выбор подходящей модели зависит от характеристик данных и задачи. Некоторые популярные методы включают линейную регрессию, решающие деревья, метод опорных векторов, нейронные сети и ансамбли моделей.
4. Препроцессинг данных: Перед обучением модели необходимо выполнить препроцессинг данных. Это может включать масштабирование признаков, заполнение пропущенных значений, удаление выбросов и кодирование категориальных переменных.
5. Разделение данных на обучающую и тестовую выборки: Чтобы оценить производительность модели, размеченные данные обычно разделяются на обучающую и тестовую выборки. Модель обучается на обучающей выборке, а затем ее производительность проверяется на тестовой выборке, которая не использовалась в процессе обучения.

Обучение моделей машинного обучения на размеченных данных требует внимательного анализа данных, правильного выбора модели и обработки данных. Правильно проведенное обучение на размеченных данных может привести к созданию эффективных моделей для прогнозирования, классификации, кластеризации и других задач анализа данных.

Вопрос № 2. Проанализируйте виды признаков, используемых в машинном обучении.

В машинном обучении признаки (фичи) представляют собой характеристики или переменные, которые описывают данные и используются моделью для принятия решений или предсказания результатов. Вот некоторые из распространенных видов признаков, которые широко используются в машинном обучении:

1. Числовые признаки: Это числовые значения, такие как возраст, доход, площадь и т. д. Числовые признаки могут быть непрерывными, когда значения лежат в некотором диапазоне, или дискретными, когда значения представляют собой отдельные категории.
2. Категориальные признаки: Они представляются категориями или метками, например, пол, страна проживания, цвет и т. д. Категориальные признаки могут быть представлены в виде текста, числовых кодов или в виде бинарных переменных (например, 0 или 1) с использованием метода кодирования, такого как one-hot encoding.
3. Порядковые признаки: Это признаки, которые имеют относительный порядок, например, оценки (низкий, средний, высокий) или уровни образования (начальное, среднее, высшее). Для обработки порядковых признаков может использоваться специальное кодирование или преобразование в числовые значения.
4. Временные ряды: Это признаки, которые отражаются во времени, например, температура, цены акций, продажи и т. д. Анализ временных рядов требует специальных методов, таких как авторегрессия (AR), скользящее среднее (MA), а также моделей ARIMA, SARIMA, LSTM и других.
5. Текстовые признаки: Это признаки, представленные текстом или последовательностями символов, например, отзывы, сообщения, документы. Для работы с текстовыми признаками применяются методы обработки естественного языка (Natural Language Processing, NLP), такие как токенизация, векторизация (например, TF-IDF или Word2Vec) и моделирование тем.
6. Изображения и видео: Визуальные данные, такие как изображения и видео, также могут быть использованы в машинном обучении.

Вопрос № 3. Рассмотрите возможности использования линейных моделей в задачах регрессии.

Линейные модели являются широко используемым инструментом в задачах регрессии. Они имеют ряд преимуществ и могут быть эффективными в различных сценариях. Вот несколько возможностей использования линейных моделей в задачах регрессии:

1. Простота и интерпретируемость: Линейные модели обладают простой структурой и легко интерпретируются. Коэффициенты модели предоставляют информацию о влиянии каждого признака на целевую переменную, что может быть полезным для понимания важности признаков и взаимодействий между ними.
2. Эффективность в больших данных: Линейные модели обладают хорошей масштабируемостью и могут быть эффективно применены к большим объемам данных. Они обычно требуют меньше вычислительных ресурсов и времени для обучения и прогнозирования, поэтому могут быть полезны в ситуациях с ограниченными ресурсами.
3. Регуляризация и управление переобучением: Линейные модели позволяют применять методы регуляризации, такие как L1 (Lasso) и L2 (Ridge), для управления переобучением и улучшения обобщающей способности модели. Это особенно полезно, когда у нас есть большое количество признаков или присутствует мультиколлинеарность.
4. Предсказание с линейной зависимостью: Если данные имеют линейную зависимость между признаками и целевой переменной, то линейная модель может быть эффективным инструментом для предсказания. Она может обнаружить и моделировать линейные связи, что особенно полезно в случаях, когда другие методы машинного обучения могут страдать от необходимости в большем объеме данных или сложных настройках.
5. Базовая модель для сравнения: Линейные модели могут служить базовыми моделями для сравнения с более сложными алгоритмами машинного обучения. Они могут помочь определить, насколько хорошо работают более сложные модели в сравнении с простыми линейными подходами и помочь в выборе наиболее подходящих.

Вопрос № 4. Рассмотрите возможности использования линейных моделей в задачах классификации.

Линейные модели также широко применяются в задачах классификации и предлагают несколько возможностей:

1. Логистическая регрессия: Логистическая регрессия является одним из популярных методов бинарной классификации. Она использует линейную комбинацию признаков и применяет логистическую функцию для прогнозирования вероятности отнесения объекта к определенному классу. Логистическая регрессия может быть расширена на мультиклассовую классификацию с помощью методов, таких как One-vs-Rest или softmax.
2. Линейный дискриминантный анализ (LDA): LDA - это метод классификации, который стремится найти линейные комбинации признаков, наиболее эффективно разделяющие классы. Он основан на оценке плотностей классов и предполагает, что признаки распределены нормально. LDA может использоваться для бинарной и мультиклассовой классификации.
3. Метод опорных векторов (SVM): SVM - это метод классификации, который строит оптимальную гиперплоскость, разделяющую классы. В линейном случае, SVM ищет линейную границу между классами, максимизируя расстояние между границей и ближайшими образцами каждого класса. SVM может использоваться для бинарной и мультиклассовой классификации.
4. Регуляризация и управление переобучением: Линейные модели в классификации также позволяют применять методы регуляризации, такие как L1 (Lasso) и L2 (Ridge), для контроля переобучения. Регуляризация помогает предотвратить слишком большие веса признаков и повышает обобщающую способность модели.
5. Базовая модель для сравнения: Линейные модели могут служить базовыми моделями для сравнения с более сложными алгоритмами классификации. Они могут быть полезны для оценки производительности более сложных моделей и для быстрой проверки гипотез, основанных на линейной связи между признаками и классами.

Линейные модели в классификации имеют свои ограничения, особенно в случаях, когда данные имеют сложную структуру.

Вопрос № 5. Задача регрессии - предсказание значений непрерывной целевой переменной

Основная цель состоит в построении модели, которая устанавливает связь между входными признаками и соответствующими значениями целевой переменной.

При решении задачи регрессии используются различные алгоритмы и модели. Некоторые из наиболее популярных методов включают:

1. Линейная регрессия: Линейная регрессия моделирует зависимость между входными признаками и целевой переменной с помощью линейной функции. Модель пытается найти наилучшую подгонку линии (или гиперплоскости в многомерном случае), минимизируя сумму квадратов ошибок.
2. Решающие деревья: Решающие деревья разбивают пространство признаков на более мелкие регионы, основываясь на пороговых значениях признаков. Каждый листовой узел дерева содержит прогнозное значение для регрессии. Решающие деревья могут быть расширены до ансамблей, таких как случайный лес или градиентный бустинг.
3. Метод опорных векторов (SVM): SVM в регрессии строит гиперплоскость с наибольшим отступом между образцами и целевыми значениями. Цель состоит в поиске оптимальной гиперплоскости, минимизирующей ошибки предсказания.
4. Метод ближайших соседей (K-NN): K-NN предсказывает значения на основе ближайших соседей в пространстве признаков. При данном методе для нового образца вычисляются ближайшие к нему обучающие образцы, и их значения используются для предсказания целевой переменной.
5. Нейронные сети: Нейронные сети представляют сложные модели, которые могут моделировать нелинейные зависимости между признаками и целевой переменной. Они состоят из множества взаимосвязанных узлов (нейронов), которые передают и обрабатывают информацию.

Вопрос № 6. Стохастический градиентный спуск

Стохастический градиентный спуск (Stochastic Gradient Descent, SGD) - это оптимизационный алгоритм, широко используемый для обучения моделей машинного обучения, включая задачи регрессии. Он является итеративным алгоритмом, который приближенно находит минимум (или максимум) функции стоимости, обновляя параметры модели в направлении, противоположном градиенту функции стоимости.

Основная идея стохастического градиентного спуска заключается в использовании случайно выбранных подмножеств обучающих данных (называемых пакетами или мини-пакетами) для оценки градиента функции стоимости и обновления параметров модели. В отличие от полного градиентного спуска, который вычисляет градиент по всем образцам данных, SGD вычисляет градиент по одному или нескольким случайно выбранным образцам одновременно.

Преимущества стохастического градиентного спуска включают:

1. Эффективность на больших данных: SGD работает эффективно на больших объемах данных, поскольку требует вычисления градиента только на небольшом подмножестве данных.
2. Скорость обучения: Благодаря обновлению параметров на каждой итерации, SGD может быстро сойтись к оптимальным значениям параметров модели.
3. Способность к онлайн-обучению: SGD может быть использован для обучения модели по мере поступления новых данных (онлайн-обучение), что делает его полезным в ситуациях, когда данные поступают потоком или обновляются со временем.
4. Возможность обработки шумных данных: Использование случайных подмножеств данных в SGD помогает справиться с шумом или выбросами в данных, поскольку случайные выборки могут сгладить эти аномалии.

5. Параллелизация: Итерации SGD могут быть выполнены независимо для разных подмножеств данных, что позволяет его параллелизировать на многопроцессорных системах и ускорить процесс обучения.
Однако SGD также имеет свои ограничения, такие как большая чувствительность к выбору скорости обучения и сложность подбора оптимальных гиперпараметров.

Вопрос № 7. K ближайших соседей
K ближайших соседей (K-Nearest Neighbors, K-NN) - это алгоритм машинного обучения, используемый в задачах классификации и регрессии. В контексте задачи регрессии, K-NN используется для предсказания значений непрерывной целевой переменной на основе ближайших соседей в пространстве признаков.
Основная идея K-NN заключается в следующем:
1. Выберите значение K, которое представляет количество ближайших соседей, учитываемых при прогнозировании.
2. Оцените расстояние (например, евклидово расстояние или другие метрики) между целевым образцом и всеми обучающими образцами в пространстве признаков.
3. Выберите K образцов с наименьшим расстоянием до целевого образца.
4. Для регрессии вычислите среднее значение целевой переменной (для регрессии) или взвешенное среднее значение (с использованием весов, основанных на расстоянии), используя выбранные K образцов. Это значение будет предсказанием для целевого образца.
Преимущества K-NN включают:
1. Простота реализации и понимания: K-NN является относительно простым алгоритмом, который не требует сложных предположений или настройки параметров.
2. Гибкость: K-NN может быть применен к различным типам данных и нечувствителен к форме распределения признаков.
3. Устойчивость к выбросам: За счет учета K ближайших соседей, K-NN менее подвержен влиянию выбросов или шума в данных.

Однако у K-NN есть некоторые ограничения:
1. Вычислительная сложность: При большом количестве образцов в обучающем наборе поиск ближайших соседей может быть вычислительно сложным.
2. Зависимость от выбора K: Выбор правильного значения K может быть нетривиальной задачей. Слишком маленькое K может привести к переобучению, а слишком большое K может привести к недообучению модели.
3. Чувствительность к масштабированию.

Вопрос № 8. Простая линейная регрессия
Простая линейная регрессия - это модель машинного обучения, которая устанавливает линейную зависимость между одним входным признаком (предиктором) и непрерывной целевой переменной. Она представляет собой простейшую форму линейной регрессии, где предсказываемая переменная зависит только от одного предиктора.
Математически простая линейная регрессия может быть представлена следующим уравнением:
$y = \beta_0 + \beta_1 * x$
где:
- y - целевая переменная, которую мы хотим предсказать,
- x - входной признак (предиктор),
- β_0 - коэффициент смещения (пересечение с осью y),
- β_1 - коэффициент наклона прямой (угловой коэффициент).
Цель состоит в определении наилучших значений для коэффициентов β_0 и β_1, которые наилучшим образом соответствуют зависимости между предиктором и целевой переменной. Для этого используется метод наименьших квадратов (МНК), который минимизирует сумму квадратов ошибок между фактическими значениями целевой переменной и предсказанными значениями модели.
Преимущества простой линейной регрессии включают:

1. Простота интерпретации: Модель простой линейной регрессии имеет простую геометрическую интерпретацию. Коэффициент наклона β_1 представляет собой изменение в целевой переменной, связанное с изменением предиктора на единицу.
2. Вычислительная эффективность: Построение модели простой линейной регрессии и вычисление коэффициентов может быть выполнено аналитически, что делает этот метод вычислительно эффективным.
3. Базовая модель для сравнения: Простая линейная регрессия может использоваться в качестве базовой модели для сравнения с более сложными алгоритмами регрессии. Она может помочь определить, достигается ли улучшение модели при добавлении дополнительных признаков или изменении модели.
Однако простая линейная регрессия также имеет свои ограничения:
1. Линейная зависимость.
2. Предположение о линейности: Простая линейная регрессия предполагает, что отношение между предиктором и целевой переменной является линейным. Если связь между ними является нелинейной, то простая линейная регрессия может быть недостаточно гибкой для точного моделирования данных.
3. Одномерность: Простая линейная регрессия ограничена одним предиктором, что может быть недостаточным для учета сложных взаимодействий или зависимостей в данных. В реальных задачах часто требуется учет нескольких признаков, и в таких случаях простая линейная регрессия может не быть адекватной моделью.
4. Чувствительность к выбросам: Простая линейная регрессия может быть чувствительна к наличию выбросов в данных, которые могут сильно влиять на оценку коэффициентов модели и приводить к неправильным предсказаниям.
5. Нарушение предположений: Простая линейная регрессия имеет предположения о нормальности ошибок, гомоскедастичности (одинаковой дисперсии) и отсутствии автокорреляции в ошибках. Если эти предположения не выполняются, то результаты модели могут быть ненадежными.
В случаях, когда ограничения простой линейной регрессии не соблюдаются или требуется учет нескольких предикторов, можно использовать более сложные модели регрессии, такие как множественная линейная регрессия, полиномиальная регрессия, регрессия с регуляризацией (например, лассо или ридж регрессия) или непараметрические методы, которые могут обрабатывать более сложные зависимости в данных.

Вопрос № 9. Логистическая регрессия.
Логистическая регрессия - это алгоритм машинного обучения, который используется в задачах бинарной классификации, где цель состоит в прогнозировании вероятности принадлежности объекта к определенному классу.
Основная идея логистической регрессии заключается в моделировании логистической функции (также известной как сигмоидная функция) для предсказания вероятности класса. Модель принимает на вход значения признаков и выдает вероятность принадлежности к положительному классу (обычно обозначаемому как класс 1).
Математически логистическая регрессия может быть представлена следующим уравнением:
$P(y=1|x) = 1 / (1 + \exp(-z))$
где:
- $P(y=1|x)$ - вероятность принадлежности к положительному классу для заданного наблюдения с признаками x,
- z - линейная комбинация весов признаков и их значений.
Процесс обучения логистической регрессии заключается в оценке оптимальных значений весов модели путем оптимизации функции потерь. Часто используется функция потерь Log Loss или Cross-Entropy Loss, которая минимизирует разницу между предсказанными вероятностями и фактическими метками классов.
Преимущества логистической регрессии включают:

1. Интерпретируемость: Веса модели могут быть интерпретированы для понимания влияния каждого признака на вероятность принадлежности к классу.
2. Эффективность: Логистическая регрессия имеет высокую вычислительную эффективность и обладает хорошей масштабируемостью для больших наборов данных.
3. Вероятностная интерпретация: Модель предсказывает вероятность принадлежности к классу, что может быть полезно для принятия решений с учетом неопределенности.
Однако логистическая регрессия также имеет некоторые ограничения:
1. Линейная разделимость: Логистическая регрессия предполагает линейную разделимость между классами. Если классы не могут быть хорошо разделены линейной границей, то логистическая регрессия может быть недостаточно точной.
2. Чувствительность к выбросам: Логистическая регрессия может быть чувствительна к выбросам в данных, которые могут исказить оценку весов и привести к неправильным предсказаниям. Необходимо уделить внимание предварительной обработке данных и устойчивости модели к выбросам.
3. Линейная зависимость: Логистическая регрессия предполагает линейную зависимость между признаками и логарифмом шансов классификации. Если связь между признаками и целевой переменной является нелинейной, то может потребоваться использование полиномиальных признаков или других методов для учета нелинейных зависимостей.
4. Проблема мультиколлинеарности: Мультиколлинеарность возникает, когда между признаками существует высокая корреляция. Это может привести к нестабильности оценок весов и затруднить интерпретацию модели. В таких случаях может потребоваться применение методов регуляризации или удаление избыточных признаков.
5. Ограниченность для многоклассовой классификации: Логистическая регрессия по умолчанию предназначена для бинарной классификации. Для решения задач многоклассовой классификации требуется расширение модели, например, с использованием методов "один против всех" или "один против одного".
Не смотря на эти ограничения, логистическая регрессия остается полезным и широко применяемым методом для задач бинарной классификации, особенно когда данные линейно разделимы и имеют относительно небольшое количество признаков.

Вопрос № 10. Деревья принятия решений.
Деревья принятия решений - это графический модельный метод машинного обучения, который используется для решения задач классификации и регрессии. Они представляют собой структуру в виде дерева, где каждый узел представляет признак, каждое ребро - возможное значение этого признака, а каждый лист - предсказанное значение класса или регрессионное значение.
Процесс построения дерева принятия решений основывается на разбиении набора данных на подмножества на основе значений признаков. Оптимальные разбиения строятся с учетом различных критериев, таких как прирост информации (Information Gain), индекс Джини (Gini Index) или энтропия (Entropy), чтобы максимизировать чистоту (или минимизировать неопределенность) классов в каждом разбиении.
Преимущества деревьев принятия решений включают:
1. Интерпретируемость: Деревья принятия решений предоставляют понятное представление, которое можно легко интерпретировать. Решения, принимаемые на основе структуры дерева, могут быть понятными для экспертов и неспециалистов.
2. Универсальность: Деревья принятия решений могут быть применены к различным типам данных, включая категориальные и числовые признаки. Они также могут обрабатывать как бинарные, так и многоклассовые задачи классификации, а также задачи регрессии.
3. Не требуют предварительной обработки данных: Деревья принятия решений не требуют масштабирования или нормализации признаков и могут хорошо работать с отсутствующими значениями.
Однако у деревьев принятия решений также есть некоторые ограничения:
1. Склонность к переобучению: Большие деревья могут сильно подстраиваться под обучающие данные и переобучаться, что может привести к плохим результатам на новых данных. Для борьбы с переобучением могут быть применены методы обрезки деревьев или

ансамблирования, такие как случайный лес (Random Forest) или градиентный бустинг (Gradient Boosting).
2. Недостаточная обобщающая способность: Деревья принятия решений могут иметь тенденцию строить слишком сложные модели, которые хорошо соответствуют обучающим данным, но плохо обобщаются на новые данные. Это может быть особенно проблематично, если данные содержат шум или недостаточное количество образцов.
3. Жесткость разбиения: Деревья принятия решений могут создавать жесткие границы разбиения признаков, что может быть нежелательным в случае, когда классы не могут быть линейно разделены. Для более сложных зависимостей в данных может потребоваться использование ансамблей деревьев или других моделей.
4. Неустойчивость к изменениям данных: Деревья принятия решений могут быть чувствительны к небольшим изменениям в обучающих данных и могут приводить к значительным изменениям в структуре дерева. Это означает, что небольшие изменения в данных могут приводить к драматическим изменениям в предсказаниях модели.
5. Проблема поиска оптимальной структуры: Построение оптимального дерева принятия решений является вычислительно сложной задачей. В некоторых случаях построение дерева может потребовать значительных вычислительных ресурсов или времени.
В целом, деревья принятия решений представляют мощный инструмент для моделирования и интерпретации данных, но их использование требует осторожности и применения соответствующих методов регуляризации и ансамблирования для улучшения обобщающей способности и стабильности модели.

Вопрос №1.

1.1 Задайте функцию, которая не принимает никаких аргументов и просто выводит на экран строку: It's my first function
В конце программы вызовите эту функцию.

```
def my_function():
    print("It's my first function")
# вызов функции
my_function()
```

1.2 Имеется список с оценками студента:
m = [2, 3, 5, 5, 2, 2, 3, 3, 4, 5, 4, 4]
Необходимо с помощью срезов выбрать элементы с 3-го по 7-й (включительно) и вывести их на экран в обратном порядке.

```
m = [2, 3, 5, 5, 2, 2, 3, 3, 4, 5, 4, 4]
# выбираем элементы с 3-го по 7-й (включительно) с помощью среза
selected_elements = m[2:7]
# выводим выбранные элементы на экран в обратном порядке
print(selected_elements[::-1])
```

*В результате выполнения этого кода на экран будет выведен список **[3, 2, 5, 5, 3]**, который содержит элементы с 3-го по 7-й (включительно) в обратном порядке.*

Вопрос №2.

2.1 Вводятся целые числа в одну строчку через пробел. Необходимо преобразовать их в список lst , затем, удалить последнее значение и если оно нечетное, то в список (в конец) добавить True, иначе - False. Отобразить полученный список на экране командой:
print(*lst)
Реализовать программу без использования условного оператора.

```
# вводим числа через пробел и преобразуем их в список
lst = list(map(int, input().split()))
# определяем четность последнего значения
last_element_is_odd = lst[-1] % 2 == 1
# удаляем последнее значение из списка
lst.pop()
# добавляем в список значение, соответствующее четности последнего элемента
lst.append(last_element_is_odd)
# выводим список на экран
print(*lst)
```

4 5 3 2
4 5 3 False

*В этом примере мы сначала проверяем длину списка с помощью функции **len()**. Если в списке есть как минимум два элемента, мы выполняем операцию **pop()** и добавляем новый элемент в конец списка. Если же в списке менее двух элементов, мы не выполняем никаких операций.*

2.2 На вход программы поступают данные в виде набора строк в формате:
ключ1=значение1
ключ2=значение2
...
ключN=значениеN
Ключами здесь выступают целые числа (см. пример ниже). Необходимо их преобразовать в словарь d (без использования функции dict()) и вывести его на экран командой:
print(*sorted(d.items()))
P. S. Для считывания списка целиком в программе уже записаны начальные строчки.

```
# начальные строки программы
input_data = ['1=apple', '2=banana', '3=cherry']
# создаем пустой словарь
d = {}
# проходим по каждой строке в списке input_data
for line in input_data:
    # разделяем строку на ключ и значение
    key, value = line.split('=')
    # преобразуем ключ в целое число
    key = int(key)
    # добавляем ключ и значение в словарь
    d[key] = value
# выводим отсортированный словарь на экран
print(*sorted(d.items()))
(1, 'apple') (2, 'banana') (3, 'cherry')
```

*Здесь мы сначала создаем пустой словарь **d**. Затем мы проходим по каждой строке в списке **input_data**. В каждой строке мы разделяем ключ и значение, используя разделитель '='. Затем мы преобразуем ключ в целое число с помощью функции **int()**. Наконец, мы добавляем ключ и значение в словарь **d** с помощью операции присваивания. После того, как мы обработали все строки в списке **input_data**, мы выводим отсортированный словарь на экран, используя метод **items()** для получения пар ключ-значение и функцию **sorted()** для их сортировки. Функция **print()** с аргументом **** развертывает пары ключ-значение в отдельные аргументы, разделенные пробелами.*

Вопрос №3.

3.1 Вводятся оценки студента (числа от 2 до 5) в одну строку через пробел. Необходимо определить количество двоек и вывести это значение на экран.

```
# вводим оценки студента через пробел
grades = list(map(int, input().split()))
# определяем количество двоек
count_of_twos = grades.count(2)
# выводим количество двоек на экран
print(count_of_twos)
1 2 2 3
2
```

*В этом примере мы вводим оценки студента через пробел и преобразуем их в список **grades** с помощью функции **map()** и **list()**. Затем мы используем метод **count()** для подсчета количества вхождений числа 2 в списке **grades**. Полученное значение сохраняем в переменную **count_of_twos**. Наконец, мы выводим количество двоек на экран с помощью функции **print()**.*

3.2 Вводятся данные в формате ключ=значение в одну строчку через пробел. Необходимо на их основе создать словарь, затем проверить, существуют ли в нем ключи со значениями: 'house', 'True' и '5' (все ключи - строки). Если все они существуют, то вывести на экран ДА, иначе - НЕТ.

```
# вводим данные в формате ключ=значение через пробел
data = input().split()
# создаем пустой словарь
dictionary = {}
# заполняем словарь значениями из введенных данных
for item in data:
    key, value = item.split('=')
```

```
    dictionary[key] = value
# проверяем наличие требуемых ключей в словаре
key1_exists = 'house' in dictionary
key2_exists = 'True' in dictionary
key3_exists = '5' in dictionary
# выводим результат на экран
if key1_exists and key2_exists and key3_exists:
    print('ДА')
else:
    print('НЕТ')
```

house=01 True=02 5=03
ДА

В этом примере мы сначала вводим данные в формате "ключ=значение" через пробел и разделяем их с помощью метода split(), сохраняя результат в список data. Затем мы создаем пустой словарь dictionary.
Затем мы проходим по каждому элементу списка data и разделяем его на ключ и значение с помощью метода split('='). Затем мы добавляем полученные пары ключ-значение в словарь dictionary.
Далее мы проверяем наличие требуемых ключей 'house', 'True' и '5' в словаре с помощью оператора in. Результат проверки сохраняем в отдельные переменные key1_exists, key2_exists и key3_exists.
Наконец, мы выводим результат на экран в зависимости от наличия всех требуемых ключей в словаре с помощью условного оператора if-else. Если все ключи существуют, выводим 'ДА', иначе выводим 'НЕТ'.

Вопрос №4.
4.1 Вводятся данные в формате ключ=значение в одну строчку через пробел. Необходимо на их основе создать словарь d, затем удалить из этого словаря ключи 'False' и '3', если они существуют. Ключами и значениями словаря являются строки. Вывести полученный словарь на экран командой: print(*sorted(d.items()))

```
# вводим данные в формате ключ=значение через пробел
data = input().split()
# создаем пустой словарь
d = {}
# заполняем словарь значениями из введенных данных
for item in data:
    key, value = item.split('=')
    d[key] = value
# удаляем ключи 'False' и '3', если они существуют
if 'False' in d:
    del d['False']
if '3' in d:
    del d['3']
# выводим полученный словарь на экран
print(*sorted(d.items()))
```

a=01 False=02 3=03 4=04
('4', '04') ('a', '01')

*В этом примере мы сначала вводим данные в формате "ключ=значение" через пробел и разделяем их с помощью метода **split()**, сохраняя результат в список **data**. Затем мы создаем пустой словарь **d**.*
*Затем мы проходим по каждому элементу списка **data** и разделяем его на ключ и значение с помощью метода **split('=')**. Затем мы добавляем полученные пары ключ-значение в словарь **d**.*

*Далее мы проверяем наличие ключей 'False' и '3' в словаре **d** с помощью оператора **in**. Если они существуют, мы используем оператор **del** для удаления соответствующих ключей из словаря.*
*Наконец, мы выводим полученный словарь на экран, сортируя его пары ключ-значение с помощью функции **sorted()** и разворачивая их с помощью оператора ***** в отдельные аргументы для функции **print()**.*

4.2 Вводятся номера телефонов в одну строчку через пробел с разными кодами стран: +7, +6, +2, +4 и т.д. Необходимо составить словарь d, где ключи - это коды +7, +6, +2 и т.п., а значения - список номеров (следующих в том же порядке, что и во входной строке) с соответствующими кодами. Полученный словарь вывести командой:
print(*sorted(d.items()))

```
# вводим номера телефонов в одну строку через пробел
phone_numbers = input().split()
# создаем пустой словарь
d = {}
# проходим по каждому номеру телефона
for number in phone_numbers:
    # извлекаем код страны из номера
    country_code = number[:2]
    # если код страны уже есть в словаре, добавляем номер в соответствующий список
    if country_code in d:
        d[country_code].append(number)
    # иначе создаем новую запись в словаре с пустым списком номеров
    else:
        d[country_code] = [number]
# выводим полученный словарь на экран
print(*sorted(d.items()))
```

```
+77712110729
('+7', ['+77712110729'])
```

*В этом примере мы сначала вводим номера телефонов в одну строку через пробел и разделяем их с помощью метода **split()**, сохраняя результат в список **phone_numbers**.*
*Затем мы создаем пустой словарь **d**.*
*Затем мы проходим по каждому номеру телефона в списке **phone_numbers**. Для каждого номера мы извлекаем первые два символа, которые представляют код страны, с помощью среза **number[:2]**. Затем мы проверяем, есть ли уже такой код страны в словаре **d**. Если код страны уже есть, мы добавляем текущий номер в соответствующий список номеров в словаре. Если же код страны отсутствует в словаре, мы создаем новую запись с ключом в виде кода страны и значением в виде пустого списка, и добавляем текущий номер в этот список.*
*Наконец, мы выводим полученный словарь на экран, сортируя его пары ключ-значение с помощью функции **sorted()** и разворачивая их с помощью оператора ***** в отдельные аргументы для функции **print()**.*

Вопрос №5.
5.1 Пользователь вводит в цикле целые положительные числа, пока не введет число 0. Для каждого числа вычисляется квадратный корень (с точностью до сотых) и значение выводится на экран (в столбик). С помощью словаря выполните кэширование данных так, чтобы при повторном вводе того же самого числа результат не вычислялся, а бралось ранее вычисленное значение из словаря. При этом на экране должно выводиться: значение из кэша: <число>

```
import math

# создаем пустой словарь для кэширования результатов
```

```
cache = {}

while True:
    # вводим число
    number = int(input("Введите число (0 для выхода): "))

    # проверяем, является ли число нулем
    if number == 0:
        break

    # проверяем, есть ли значение в кэше
    if number in cache:
        print("Значение из кэша:", cache[number])
    else:
        # вычисляем квадратный корень с точностью до сотых
        sqrt_value = round(math.sqrt(number), 2)
        # сохраняем результат в кэше
        cache[number] = sqrt_value
        print(sqrt_value)
Введите число (0 для выхода):  1
1.0
Введите число (0 для выхода):  2
1.41
Введите число (0 для выхода):  2
Значение из кэша: 1.41
Введите число (0 для выхода):  9
3.0
Введите число (0 для выхода):  0
```

*В этом примере мы создаем пустой словарь **cache** для кэширования результатов. Затем мы входим в цикл, который будет выполняться до тех пор, пока пользователь не введет число 0.*
*В каждой итерации цикла мы сначала вводим число с помощью функции **input()**, преобразуем его в целое число с помощью **int()**, и сохраняем результат в переменную **number**.*
*Затем мы проверяем, есть ли значение в словаре **cache** для данного числа. Если значение уже есть в кэше, мы выводим сообщение "Значение из кэша:" и выводим соответствующее значение из словаря **cache**. Если значение отсутствует в кэше, мы вычисляем квадратный корень с помощью функции **math.sqrt()** и округляем его до двух знаков после запятой с помощью функции **round()**. Затем сохраняем результат в кэше, используя число как ключ и вычисленное значение как значение. После этого мы выводим вычисленный квадратный корень на экран.*
*Таким образом, при повторном вводе того же числа результат будет браться из словаря **cache**, что обеспечивает кэширование данных.*

5.2 Имеется закодированная строка с помощью азбуки Морзе. Коды разделены между собой пробелом. Необходимо ее раскодировать, используя азбуку Морзе из предыдущего занятия. Полученное сообщение (строку) вывести на экран.

```
# Словарь с кодами Морзе
morse_code = {
    '.-': 'A', '-...': 'B', '-.-.': 'C', '-..': 'D', '.': 'E',
    '..-.': 'F', '--.': 'G', '....': 'H', '..': 'I', '.---': 'J',
    '-.-': 'K', '.-..': 'L', '--': 'M', '-.': 'N', '---': 'O',
    '.--.': 'P', '--.-': 'Q', '.-.': 'R', '...': 'S', '-': 'T',
    '..-': 'U', '...-': 'V', '.--': 'W', '-..-': 'X', '-.--': 'Y',
```

```
    '--..': 'Z', '.----': '1', '..---': '2', '...--': '3', '....-': '4',
    '.....': '5', '-....': '6', '--...': '7', '---..': '8', '----.': '9',
    '-----': '0', '--..--': ',', '.-.-.-': '.', '..--..': '?', '-..-.': '/',
    '-....-': '-', '-.--.': '(', '-.--.-': ')', '.-.-.': '+', '-...-': '=',
    '.-..-.': '"', '.--.-.': '@', '...---...': 'SOS'
}
# Закодированная строка
encoded_string = input("Введите закодированную строку: ")
# Разделяем закодированную строку на коды символов
codes = encoded_string.split(' ')
# Раскодированная строка
decoded_string = ''
# Проходим по каждому коду символа и добавляем соответствующий символ в раскодированную строку
for code in codes:
    if code in morse_code:
        decoded_string += morse_code[code]
    else:
        # Если код не найден в словаре, добавляем пробел
        decoded_string += ' '
# Выводим раскодированную строку на экран
print(decoded_string)
Введите закодированную строку:  .-. .. ... .--.
RISP
```

*Программа предлагает ввести закодированную строку и сохраняет ее в переменной **encoded_string**. Затем мы разделяем закодированную строку на отдельные коды символов с помощью метода **split(' ')**, сохраняя результат в списке **codes**.*
*Затем мы создаем пустую строку **decoded_string**, которая будет содержать раскодированное сообщение.*
*Затем мы проходим по каждому коду символа в списке **codes** и проверяем, есть ли этот код в словаре **morse_code**. Если код найден, мы добавляем соответствующий символ в строку **decoded_string**. Если код не найден в словаре, мы добавляем пробел*

Вопрос №6.
6.1 В список: a = [5.4, 6.7, 10.4]
добавить в конец вложенный список со значениями, вводимыми в программу (целые числа вводятся в строчку через пробел). Результирующий список lst вывести на экран командой: print(lst)

```
a = [5.4, 6.7, 10.4]
# Ввод чисел в одну строку
numbers = input("Введите целые числа через пробел: ")
# Разделяем введенные числа и преобразуем их в список целых чисел
numbers_list = list(map(int, numbers.split()))
# Добавляем список чисел в конец списка "a"
a.append(numbers_list)
# Выводим результирующий список на экран
print(a)
Введите целые числа через пробел:  1 2 45 896
[5.4, 6.7, 10.4, [1, 2, 45, 896]]
```

В этом решении мы создаем список a с исходными значениями. Затем мы запрашиваем у пользователя ввод целых чисел в одну строку и сохраняем их в переменной numbers. Мы разделяем введенные числа с помощью метода split(), преобразуем их в список целых чисел с помощью функции map(), и сохраняем результат в переменной numbers_list.

Затем мы добавляем список numbers_list в конец списка a с помощью метода append(). И, наконец, мы выводим результирующий список a на экран с помощью функции print().

6.2 Имеется кортеж: t = (3.4, -56.7)
Вводится последовательность целых чисел в одну строчку через пробел. Необходимо их добавить в кортеж t. Результат вывести на экран командой: print(t)

```
t = (3.4, -56.7)
# Ввод чисел в одну строку
numbers = input("Введите целые числа через пробел: ")
# Разделяем введенные числа и преобразуем их в список целых чисел
numbers_list = list(map(int, numbers.split()))
# Создаем новый кортеж, объединяя существующий кортеж t с новыми числами
new_t = t + tuple(numbers_list)
# Выводим результат на экран
print(new_t)
Введите целые числа через пробел:  1 2 22
(3.4, -56.7, 1, 2, 22)
```

В этом решении мы создаем кортеж t с исходными значениями. Затем мы запрашиваем у пользователя ввод целых чисел в одну строку и сохраняем их в переменной numbers. Мы разделяем введенные числа с помощью метода split(), преобразуем их в список целых чисел с помощью функции map(), и сохраняем результат в переменной numbers_list.
Затем мы создаем новый кортеж new_t, объединяя существующий кортеж t с новыми числами numbers_list с помощью оператора +. И, наконец, мы выводим результат на экран с помощью функции print().

Вопрос №7.
7.1 Вводятся целые числа в одну строку через пробел. На их основе формируется кортеж. Необходимо найти и вывести все индексы неуникальных (повторяющихся) значений в этом кортеже. Результат отобразите в виде строки чисел, записанных через пробел.

7.2 Вводятся три строчки стихотворения (каждая с новой строки). Сохранить его в виде вложенного списка с разбивкой по строкам и словам (слова разделяются пробелом). Результирующий список lst вывести на экран командой: print(lst)

Вопрос №8.
8.1 Вводится матрица чисел из трех строк. В каждой строке числа разделяются пробелом. Необходимо вывести на экран последний столбец этой матрицы в виде строки из трех чисел через пробел.

```
# Ввод чисел в одну строку
numbers = input("Введите целые числа через пробел: ")
# Разделяем введенные числа и преобразуем их в список целых чисел
numbers_list = list(map(int, numbers.split()))
# Создаем кортеж из списка чисел
t = tuple(numbers_list)
# Инициализируем пустой список для хранения индексов неуникальных значений
indexes = []
# Проверяем каждое значение в кортеже на наличие дубликатов
for i in range(len(t)):
    if t.count(t[i]) > 1:
        indexes.append(i)
# Преобразуем индексы в строку чисел, разделенных пробелами
result = ' '.join(map(str, indexes))
# Выводим результат на экран
print(result)
```

Введите целые числа через пробел: 1 2 1 3 1 4
0 2 4

В этом решении мы запрашиваем у пользователя ввод целых чисел в одну строку и сохраняем их в переменной numbers. Мы разделяем введенные числа с помощью метода split(), преобразуем их в список целых чисел с помощью функции map(), и сохраняем результат в переменной numbers_list.
Затем мы создаем кортеж t из списка чисел numbers_list с помощью функции tuple().
Затем мы проходимся по каждому значению в кортеже t с помощью цикла for. Для каждого значения мы используем метод count() для подсчета количества его повторений в кортеже. Если количество повторений больше 1, мы добавляем индекс этого значения в список indexes.
Затем мы преобразуем список indexes в строку чисел, разделенных пробелами, с помощью функций map() и join(). Результат сохраняем в переменной result.
Наконец, мы выводим результат на экран с помощью функции print().

8.2 Имеется двумерный кортеж, размером 5 x 5 элементов:
t = ((1, 0, 0, 0, 0),
 (0, 1, 0, 0, 0),
 (0, 0, 1, 0, 0),
 (0, 0, 0, 1, 0),
 (0, 0, 0, 0, 1))
Вводится натуральное число N (N < 5). Необходимо на основе кортежа t сформировать новый аналогичный кортеж t2 размером N x N элементов. Результат вывести на экран в виде таблицы чисел.

```
t = ((1, 0, 0, 0, 0),
    (0, 1, 0, 0, 0),
    (0, 0, 1, 0, 0),
    (0, 0, 0, 1, 0),
    (0, 0, 0, 0, 1))
# Ввод размера нового кортежа
N = int(input("Введите размер нового кортежа (N < 5): "))
# Создание нового кортежа t2 с размером N x N
t2 = tuple(row[:N] for row in t[:N])
# Вывод нового кортежа t2 в виде таблицы
for row in t2:
   print(*row)
```
Введите размер нового кортежа (N < 5): 3
1 0 0
0 1 0
0 0 1

В этом решении мы имеем исходный двумерный кортеж t размером 5 x 5. Затем мы запрашиваем у пользователя ввод размера нового кортежа N.
Мы создаем новый кортеж t2 с помощью генератора списков и срезов. Мы используем срезы, чтобы взять первые N строк из t, а затем для каждой строки мы берем только первые N элементов. Это позволяет создать новый кортеж t2 размером N x N.
Затем мы выводим новый кортеж t2 на экран в виде таблицы чисел, используя цикл for и функцию print().

Вопрос №9.
9.1 Вводятся два вещественных числа в одну строку через пробел. Вывести на экран наибольшее из чисел. Задачу решить с помощью условного оператора.
 # Ввод двух вещественных чисел

```
numbers = input("Введите два вещественных числа через пробел: ")
# Разделяем введенные числа и преобразуем их в список вещественных чисел
numbers_list = list(map(float, numbers.split()))
# Получаем первое и второе число из списка
num1 = numbers_list[0]
num2 = numbers_list[1]
# Сравниваем числа и выводим наибольшее
if num1 > num2:
    print("Наибольшее число:", num1)
elif num2 > num1:
    print("Наибольшее число:", num2)
else:
    print("Числа равны")
Введите два вещественных числа через пробел:  1 2
Наибольшее число: 2.0
```

В этом решении мы запрашиваем у пользователя ввод двух вещественных чисел в одну строку и сохраняем их в переменной numbers. Мы разделяем введенные числа с помощью метода split(), преобразуем их в список вещественных чисел с помощью функции map(), и сохраняем результат в переменной numbers_list.
Затем мы получаем первое и второе число из списка numbers_list и сохраняем их в переменных num1 и num2 соответственно.
Далее мы используем условный оператор if-elif-else для сравнения чисел num1 и num2. Если num1 больше num2, мы выводим "Наибольшее число: num1". Если num2 больше num1, мы выводим "Наибольшее число: num2". Если числа равны, мы выводим "Числа равны".

9.2 Вводится текст в одну строку, слова разделены пробелом. Необходимо подсчитать число уникальных слов (без учета регистра) в этом тексте. Результат (число уникальных слов) вывести на экран.

```
# Ввод текста
text = input("Введите текст: ")
# Разделение текста на слова и приведение их к нижнему регистру
words = text.lower().split()
# Создание множества для хранения уникальных слов
unique_words = set(words)
# Подсчет числа уникальных слов
count = len(unique_words)
# Вывод числа уникальных слов
print("Число уникальных слов:", count)
Введите текст:  один два один три
Число уникальных слов: 3
```

В этом решении мы сначала запрашиваем у пользователя ввод текста и сохраняем его в переменной text.
Затем мы приводим текст к нижнему регистру с помощью метода lower(), чтобы не учитывать регистр при подсчете уникальных слов. Затем мы разделяем текст на слова с помощью метода split() и сохраняем их в список words.
Далее мы создаем множество unique_words, в котором будем хранить только уникальные слова. Множество автоматически удаляет дубликаты.
Затем мы подсчитываем число уникальных слов, используя функцию len() для определения размера множества unique_words, и сохраняем результат в переменной count.
Наконец, мы выводим число уникальных слов на экран с помощью функции print().

Вопрос №10.

10.1 Вводится слово. Необходимо определить, является ли это слово палиндромом (одинаково читается вперед и назад, например, АННА). Регистр букв не учитывать. Если введенное слово палиндром, на экран вывести ДА, иначе - НЕТ.

```
# Ввод слова
word = input("Введите слово: ")
# Приведение слова к нижнему регистру
word = word.lower()
# Проверка на палиндром
if word == word[::-1]:
    print("ДА")
else:
    print("НЕТ")
```

```
Введите слово:  RISP
НЕТ
```

В этом решении мы сначала запрашиваем у пользователя ввод слова и сохраняем его в переменной word.
Затем мы приводим слово к нижнему регистру с помощью метода lower(), чтобы не учитывать регистр при проверке на палиндром.
Далее мы используем условное выражение, чтобы проверить, является ли слово палиндромом. Мы сравниваем слово word с его инвертированной версией word[::-1]. Если они равны, то слово является палиндромом, и мы выводим "ДА". В противном случае выводим "НЕТ".

10.2 Вводятся два списка целых чисел каждый с новой строки (в строке наборы чисел через пробел). Необходимо выбрать и отобразить на экране уникальные числа, присутствующие в первом списке, но отсутствующие во втором. Результат выведите на экран в виде строки чисел, записанных по возрастанию через пробел.

```
# Ввод первого списка чисел
list1 = input("Введите числа первого списка через пробел: ").split()
# Ввод второго списка чисел
list2 = input("Введите числа второго списка через пробел: ").split()
# Преобразование элементов списков в целые числа
list1 = list(map(int, list1))
list2 = list(map(int, list2))
# Создание множеств из списков
set1 = set(list1)
set2 = set(list2)
# Вычисление уникальных чисел
result = sorted(set1 - set2)
# Вывод результата
print(*result)
```

```
Введите числа первого списка через пробел:  1 2 3
Введите числа второго списка через пробел:  0 1 2
3
```

В этом решении мы сначала запрашиваем у пользователя ввод первого списка чисел и сохраняем его в переменной list1. Аналогично, запрашиваем и сохраняем второй список чисел в переменной list2.
Затем мы преобразуем элементы списков list1 и list2 из строкового типа в целочисленный тип с помощью функции map() и преобразуем результат в список.
Далее мы создаем множества set1 и set2 из списков list1 и list2 соответственно.
Затем мы находим уникальные числа, присутствующие в set1, но отсутствующие в set2, с помощью операции разности множеств set1 - set2. Результат сохраняем в переменной result и сортируем его с помощью функции sorted().

*Наконец, мы выводим результат на экран с помощью функции print(), разделяя элементы пробелами с помощью оператора *.*