объясни код classification = pd.read\_csv("airlines\_task.csv")

classification.loc[:, ["Airline", "AirportFrom", "AirportTo", "Flight"]].nunique(axis=0)

classification.dropna(inplace=True)

data\_encoded = pd.get\_dummies(classification, columns=["Airline"], dtype=int).reset\_index(drop=True)

routes = pd.Series(data=(data\_encoded.AirportTo + data\_encoded.AirportFrom), name="Route")

routes

data\_encoded: pd.DataFrame = pd.concat([data\_encoded, routes], axis=1).drop(["AirportTo", "AirportFrom"], axis=1)

data\_encoded.rename({0: "Route"}, axis=1, inplace=True)

data\_encoded.Route.value\_counts()

label\_map = dict(zip(data\_encoded.Route.unique().tolist(), np.arange(0, data\_encoded.Route.unique().shape[0])))

label\_map

data\_encoded.Route = data\_encoded.Route.map(lambda x: label\_map.get(x))

classification = data\_encoded

regression = pd.read\_csv("diamonds.csv")

regression = pd.get\_dummies(regression, columns=['cut','color','clarity'])

regression = regression.sample(frac=0.1)

Этот код использует библиотеку pandas для обработки данных. В первой строке кода загружается файл airlines\_task.csv в DataFrame с именем classification. Затем, во второй строке, вычисляется количество уникальных значений для каждого из столбцов “Airline”, “AirportFrom”, “AirportTo” и “Flight”. В третьей строке удаляются все строки с пропущенными значениями.

В четвертой строке выполняется one-hot кодирование столбца “Airline” и результат сохраняется в новом DataFrame с именем data\_encoded. Затем создается новая серия routes, которая содержит конкатенацию значений столбцов “AirportTo” и “AirportFrom”. Эта серия добавляется к data\_encoded вместо столбцов “AirportTo” и “AirportFrom”.

В следующих строках выполняется переименование столбца 0 в “Route” и подсчет количества уникальных значений для этого столбца. Затем создается словарь label\_map, который отображает уникальные значения маршрутов на целые числа от 0 до количества уникальных маршрутов. Этот словарь используется для преобразования значений столбца “Route” в целые числа.

В последних строках кода загружается файл diamonds.csv в DataFrame с именем regression, выполняется one-hot кодирование для столбцов ‘cut’, ‘color’ и ‘clarity’ и выбирается случайная выборка из 10% данных.

2из30

объясни код y\_c = classification['Delay']

X\_c = classification.drop(['Delay'], axis=1)

y\_r = regression["price"]

X\_r = regression.drop(["price", 'Unnamed: 0'], axis=1)

Этот код использует библиотеку pandas для обработки данных. В первых двух строках кода создается переменная y\_c, которая содержит значения столбца ‘Delay’ из DataFrame classification, и переменная X\_c, которая содержит все столбцы из classification, кроме столбца ‘Delay’.

В следующих двух строках создается переменная y\_r, которая содержит значения столбца “price” из DataFrame regression, и переменная X\_r, которая содержит все столбцы из regression, кроме столбцов “price” и ‘Unnamed: 0’.

Таким образом, в переменных y\_c и y\_r хранятся целевые значения для задач классификации и регрессии соответственно, а в переменных X\_c и X\_r - соответствующие признаки.

3из30

объясни код X\_r\_train, X\_r\_test, y\_r\_train, y\_r\_test = train\_test\_split(X\_r, y\_r, test\_size=0.2)

X\_c\_train, X\_c\_test, y\_c\_train, y\_c\_test = train\_test\_split(X\_c, y\_c, test\_size=0.2)

Этот код использует функцию train\_test\_split из библиотеки scikit-learn для разделения данных на обучающую и тестовую выборки. В первой строке кода переменные X\_r\_train, X\_r\_test, y\_r\_train и y\_r\_test создаются путем разделения данных X\_r и y\_r на обучающую и тестовую выборки с использованием параметра test\_size=0.2, который указывает, что 20% данных должны быть использованы в качестве тестовой выборки.

Во второй строке кода аналогичным образом создаются переменные X\_c\_train, X\_c\_test, y\_c\_train и y\_c\_test путем разделения данных X\_c и y\_c на обучающую и тестовую выборки.

Таким образом, в переменных с суффиксом \_train хранятся данные для обучения моделей, а в переменных с суффиксом \_test - данные для оценки качества моделей.

4из30

объясни код vt\_r = VarianceThreshold(2)

X\_vt\_r = pd.DataFrame(vt\_r.fit\_transform(X\_r), columns=vt\_r.get\_feature\_names\_out())

X\_r.shape, X\_vt\_r.shape

X\_vt\_r

Этот код использует класс VarianceThreshold из библиотеки scikit-learn для удаления признаков с низкой дисперсией. В первой строке кода создается объект vt\_r класса VarianceThreshold с параметром threshold=2, который указывает, что признаки с дисперсией меньше 2 должны быть удалены.

Во второй строке кода создается новый DataFrame X\_vt\_r, который содержит результат преобразования данных X\_r с использованием объекта vt\_r. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта vt\_r.

В третьей строке кода выводятся размеры исходного DataFrame X\_r и преобразованного DataFrame X\_vt\_r. В четвертой строке кода выводится содержимое DataFrame X\_vt\_r.

Таким образом, этот код позволяет удалить признаки с низкой дисперсией из данных X\_r.

5из30

объясни код vt\_c = VarianceThreshold(0.1)

X\_vt\_c = vt\_c.fit\_transform(X\_c)

X\_vt\_c = pd.DataFrame(vt\_c.fit\_transform(X\_c), columns=vt\_c.get\_feature\_names\_out())

X\_c.shape, X\_vt\_c.shape

X\_vt\_c

Этот код использует класс VarianceThreshold из библиотеки scikit-learn для удаления признаков с низкой дисперсией. В первой строке кода создается объект vt\_c класса VarianceThreshold с параметром threshold=0.1, который указывает, что признаки с дисперсией меньше 0.1 должны быть удалены.

Во второй строке кода выполняется преобразование данных X\_c с использованием объекта vt\_c, но результат этого преобразования не сохраняется.

В третьей строке кода создается новый DataFrame X\_vt\_c, который содержит результат повторного преобразования данных X\_c с использованием объекта vt\_c. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта vt\_c.

В четвертой строке кода выводятся размеры исходного DataFrame X\_c и преобразованного DataFrame X\_vt\_c. В пятой строке кода выводится содержимое DataFrame X\_vt\_c.

Таким образом, этот код позволяет удалить признаки с низкой дисперсией из данных X\_c.

6из30

объясни код skb\_r = SelectKBest(k=3)

X\_skb\_r = pd.DataFrame(skb\_r.fit\_transform(X\_r, y\_r), columns=skb\_r.get\_feature\_names\_out())

X\_r.shape, X\_skb\_r.shape

X\_skb\_r

Этот код использует класс SelectKBest из библиотеки scikit-learn для выбора k лучших признаков. В первой строке кода создается объект skb\_r класса SelectKBest с параметром k=3, который указывает, что должны быть выбраны 3 лучших признака.

Во второй строке кода создается новый DataFrame X\_skb\_r, который содержит результат преобразования данных X\_r с использованием объекта skb\_r. Для этого преобразования также используются целевые значения y\_r. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта skb\_r.

В третьей строке кода выводятся размеры исходного DataFrame X\_r и преобразованного DataFrame X\_skb\_r. В четвертой строке кода выводится содержимое DataFrame X\_skb\_r.

Таким образом, этот код позволяет выбрать 3 лучших признака из данных X\_r.

7из30

объясни код skb\_c = SelectKBest(k=3)

X\_skb\_c = pd.DataFrame(skb\_c.fit\_transform(X\_c, y\_c), columns=skb\_c.get\_feature\_names\_out())

X\_c.shape, X\_skb\_c.shape

X\_skb\_c

Этот код использует класс SelectKBest из библиотеки scikit-learn для выбора k лучших признаков. В первой строке кода создается объект skb\_c класса SelectKBest с параметром k=3, который указывает, что должны быть выбраны 3 лучших признака.

Во второй строке кода создается новый DataFrame X\_skb\_c, который содержит результат преобразования данных X\_c с использованием объекта skb\_c. Для этого преобразования также используются целевые значения y\_c. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта skb\_c.

В третьей строке кода выводятся размеры исходного DataFrame X\_c и преобразованного DataFrame X\_skb\_c. В четвертой строке кода выводится содержимое DataFrame X\_skb\_c.

Таким образом, этот код позволяет выбрать 3 лучших признака из данных X\_c.

8из30

объясни код tree\_r = DecisionTreeRegressor().fit(X\_r, y\_r)

tree\_c = DecisionTreeClassifier().fit(X\_c, y\_c)

rfe\_r = RFE(estimator=tree\_r, n\_features\_to\_select=3, step=1).fit(X\_r, y\_r)

X\_rfe\_r = pd.DataFrame(rfe\_r.transform(X\_r), columns=rfe\_r.get\_feature\_names\_out())

X\_r.shape, X\_rfe\_r.shape

X\_rfe\_r

Этот код использует классы DecisionTreeRegressor и DecisionTreeClassifier из библиотеки scikit-learn для создания моделей регрессии и классификации соответственно. В первой строке кода создается объект tree\_r класса DecisionTreeRegressor и обучается на данных X\_r и y\_r. Во второй строке кода аналогичным образом создается объект tree\_c класса DecisionTreeClassifier и обучается на данных X\_c и y\_c.

В третьей строке кода создается объект rfe\_r класса RFE с параметрами estimator=tree\_r, n\_features\_to\_select=3 и step=1. Этот объект используется для выбора 3 лучших признаков из данных X\_r с использованием обученного ранее регрессора tree\_r. Объект rfe\_r обучается на данных X\_r и y\_r.

В четвертой строке кода создается новый DataFrame X\_rfe\_r, который содержит результат преобразования данных X\_r с использованием объекта rfe\_r. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта rfe\_r.

В пятой строке кода выводятся размеры исходного DataFrame X\_r и преобразованного DataFrame X\_rfe\_r. В шестой строке кода выводится содержимое DataFrame X\_rfe\_r.

Таким образом, этот код позволяет выбрать 3 лучших признака из данных X\_r с использованием обученного ранее регрессора.

9из30

объясни код rfe\_c = RFE(estimator=tree\_c, n\_features\_to\_select=3, step=1).fit(X\_c, y\_c)

X\_rfe\_c = pd.DataFrame(rfe\_c.transform(X\_c), columns=rfe\_c.get\_feature\_names\_out())

X\_c.shape, X\_rfe\_c.shape

X\_rfe\_c

Этот код использует класс RFE из библиотеки scikit-learn для выбора k лучших признаков. В первой строке кода создается объект rfe\_c класса RFE с параметрами estimator=tree\_c, n\_features\_to\_select=3 и step=1. Этот объект используется для выбора 3 лучших признаков из данных X\_c с использованием обученного ранее классификатора tree\_c. Объект rfe\_c обучается на данных X\_c и y\_c.

Во второй строке кода создается новый DataFrame X\_rfe\_c, который содержит результат преобразования данных X\_c с использованием объекта rfe\_c. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта rfe\_c.

В третьей строке кода выводятся размеры исходного DataFrame X\_c и преобразованного DataFrame X\_rfe\_c. В четвертой строке кода выводится содержимое DataFrame X\_rfe\_c.

Таким образом, этот код позволяет выбрать 3 лучших признака из данных X\_c с использованием обученного ранее классификатора.

10из30

объясни код pca\_r = PCA(n\_components=3)

X\_pca\_r = pd.DataFrame(pca\_r.fit\_transform(X\_r, y\_r), columns=pca\_r.get\_feature\_names\_out())

X\_r.shape, X\_pca\_r.shape

X\_pca\_r

Этот код использует класс PCA из библиотеки scikit-learn для выполнения метода главных компонент (PCA) на данных. В первой строке кода создается объект pca\_r класса PCA с параметром n\_components=3, который указывает, что должны быть выбраны 3 главные компоненты.

Во второй строке кода создается новый DataFrame X\_pca\_r, который содержит результат преобразования данных X\_r с использованием объекта pca\_r. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта pca\_r.

В третьей строке кода выводятся размеры исходного DataFrame X\_r и преобразованного DataFrame X\_pca\_r. В четвертой строке кода выводится содержимое DataFrame X\_pca\_r.

Таким образом, этот код позволяет выполнить метод главных компонент на данных X\_r и выбрать 3 главные компоненты.

11из30

объясни код pca\_c= PCA(n\_components=3)

X\_pca\_c = pca\_c.fit\_transform(X\_c, y\_c)

X\_pca\_c = pd.DataFrame(pca\_c.fit\_transform(X\_c, y\_c), columns=pca\_c.get\_feature\_names\_out())

X\_c.shape, X\_pca\_c.shape

X\_pca\_r

Этот код использует класс PCA из библиотеки scikit-learn для выполнения метода главных компонент (PCA) на данных. В первой строке кода создается объект pca\_c класса PCA с параметром n\_components=3, который указывает, что должны быть выбраны 3 главные компоненты.

Во второй строке кода выполняется преобразование данных X\_c с использованием объекта pca\_c, но результат этого преобразования не сохраняется.

В третьей строке кода создается новый DataFrame X\_pca\_c, который содержит результат повторного преобразования данных X\_c с использованием объекта pca\_c. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта pca\_c.

В четвертой строке кода выводятся размеры исходного DataFrame X\_c и преобразованного DataFrame X\_pca\_c. В пятой строке кода выводится содержимое DataFrame X\_pca\_r, хотя, вероятно, имелось в виду вывести содержимое DataFrame X\_pca\_c.

Таким образом, этот код позволяет выполнить метод главных компонент на данных X\_c и выбрать 3 главные компоненты.

12из30

объясни код tsne\_r = TSNE(n\_components=3)

X\_tsne\_r = tsne\_r.fit\_transform(X\_r)

X\_r.shape, X\_tsne\_r.shape

Этот код использует класс TSNE из библиотеки scikit-learn для выполнения алгоритма t-SNE на данных. В первой строке кода создается объект tsne\_r класса TSNE с параметром n\_components=3, который указывает, что должны быть выбраны 3 компоненты.

Во второй строке кода выполняется преобразование данных X\_r с использованием объекта tsne\_r и результат этого преобразования сохраняется в переменной X\_tsne\_r.

В третьей строке кода выводятся размеры исходного DataFrame X\_r и преобразованного массива X\_tsne\_r.

Таким образом, этот код позволяет выполнить алгоритм t-SNE на данных X\_r и выбрать 3 компоненты.

13из30

объясни код tsne\_c = TSNE(n\_components=3)

X\_tsne\_c = tsne\_c.fit\_transform(X\_c)

X\_c.shape, X\_tsne\_c.shape

Этот код использует класс TSNE из библиотеки scikit-learn для выполнения алгоритма t-SNE на данных. В первой строке кода создается объект tsne\_c класса TSNE с параметром n\_components=3, который указывает, что должны быть выбраны 3 компоненты.

Во второй строке кода выполняется преобразование данных X\_c с использованием объекта tsne\_c и результат этого преобразования сохраняется в переменной X\_tsne\_c.

В третьей строке кода выводятся размеры исходного DataFrame X\_c и преобразованного массива X\_tsne\_c.

Таким образом, этот код позволяет выполнить алгоритм t-SNE на данных X\_c и выбрать 3 компоненты.

14из30

объясни код isomap\_r = Isomap(n\_components=3)

X\_isomap\_r = pd.DataFrame(isomap\_r.fit\_transform(X\_r), columns=isomap\_r.get\_feature\_names\_out())

X\_r.shape, X\_isomap\_r.shape

X\_isomap\_r

Этот код использует класс Isomap из библиотеки scikit-learn для выполнения алгоритма Isomap на данных. В первой строке кода создается объект isomap\_r класса Isomap с параметром n\_components=3, который указывает, что должны быть выбраны 3 компоненты.

Во второй строке кода создается новый DataFrame X\_isomap\_r, который содержит результат преобразования данных X\_r с использованием объекта isomap\_r. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта isomap\_r.

В третьей строке кода выводятся размеры исходного DataFrame X\_r и преобразованного DataFrame X\_isomap\_r. В четвертой строке кода выводится содержимое DataFrame X\_isomap\_r.

Таким образом, этот код позволяет выполнить алгоритм Isomap на данных X\_r и выбрать 3 компоненты.

15из30

объясни код isomap\_c = Isomap(n\_components=3, max\_iter=50)

X\_isomap\_c = pd.DataFrame(isomap\_c.fit\_transform(X\_c[:5000]), columns=isomap\_c.get\_feature\_names\_out())

X\_c.shape, X\_isomap\_c.shape

X\_isomap\_c

Этот код использует класс Isomap из библиотеки scikit-learn для выполнения алгоритма Isomap на данных. В первой строке кода создается объект isomap\_c класса Isomap с параметрами n\_components=3 и max\_iter=50, которые указывают, что должны быть выбраны 3 компоненты и максимальное количество итераций алгоритма равно 50.

Во второй строке кода создается новый DataFrame X\_isomap\_c, который содержит результат преобразования первых 5000 строк данных X\_c с использованием объекта isomap\_c. Столбцы нового DataFrame имеют имена, возвращаемые методом get\_feature\_names\_out объекта isomap\_c.

В третьей строке кода выводятся размеры исходного DataFrame X\_c и преобразованного DataFrame X\_isomap\_c. В четвертой строке кода выводится содержимое DataFrame X\_isomap\_c.

Таким образом, этот код позволяет выполнить алгоритм Isomap на первых 5000 строках данных X\_c и выбрать 3 компоненты.

16из30

объясни код def test\_r(X, y):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, shuffle=False)

bag = BaggingRegressor().fit(X\_train, y\_train)

print(f'MAE: {mean\_absolute\_error(y\_test, bag.predict(X\_test))}')

def test\_c(X, y):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, shuffle=False)

bag = BaggingClassifier().fit(X\_train, y\_train)

print(accuracy\_score(y\_test, bag.predict(X\_test)))

Этот код содержит две функции: test\_r и test\_c. Функция test\_r принимает два аргумента: X и y, которые представляют собой данные для задачи регрессии. Внутри функции выполняется разделение данных на обучающую и тестовую выборки с использованием функции train\_test\_split из библиотеки scikit-learn. Параметр test\_size=0.15 указывает, что 15% данных должны быть использованы в качестве тестовой выборки, а параметр shuffle=False указывает, что данные не должны перемешиваться перед разделением.

Затем создается объект bag класса BaggingRegressor и обучается на обучающей выборке. В конце функции выводится средняя абсолютная ошибка (MAE) между истинными значениями тестовой выборки и предсказанными значениями, полученными с использованием обученного регрессора.

Функция test\_c аналогична функции test\_r, но используется для задачи классификации. Она принимает два аргумента: X и y, которые представляют собой данные для задачи классификации. Внутри функции выполняется разделение данных на обучающую и тестовую выборки таким же образом, как в функции test\_r.

Затем создается объект bag класса BaggingClassifier и обучается на обучающей выборке. В конце функции выводится точность (accuracy) между истинными метками классов тестовой выборки и предсказанными метками, полученными с использованием обученного классификатора.

17из30

объясни код print("VarianceThreshold")

test\_r(X\_vt\_r, y\_r)

print("SelectKBest")

test\_r(X\_skb\_r, y\_r)

print("RFE")

test\_r(X\_rfe\_r, y\_r)

print("PCA")

test\_r(X\_pca\_r, y\_r)

print("TSNE")

test\_r(X\_tsne\_r, y\_r)

print("Isomap")

test\_r(X\_isomap\_r, y\_r)

Этот код использует функцию test\_r, которая, вероятно, была определена ранее. Функция test\_r принимает два аргумента: X и y, которые представляют собой данные для задачи регрессии.

В коде последовательно выводятся названия методов выбора признаков и вызывается функция test\_r с соответствующими данными. Например, в первой строке кода выводится строка “VarianceThreshold”, а затем вызывается функция test\_r с аргументами X\_vt\_r и y\_r. Это означает, что функция test\_r будет протестирована на данных, полученных с использованием метода VarianceThreshold.

Аналогичным образом для каждого из методов выбора признаков выводится его название и вызывается функция test\_r с соответствующими данными.

Таким образом, этот код позволяет протестировать функцию test\_r на данных, полученных с использованием различных методов выбора признаков.

18из30

объясни код print("VarianceThreshold")

test\_c(X\_vt\_c, y\_c)

print("SelectKBest")

test\_c(X\_skb\_c, y\_c)

print("RFE")

test\_c(X\_rfe\_c, y\_c)

print("PCA")

test\_c(X\_pca\_c, y\_c)

print("Isomap")

test\_c(X\_isomap\_c, y\_c[:5000])

Этот код использует функцию test\_c, которая, вероятно, была определена ранее. Функция test\_c принимает два аргумента: X и y, которые представляют собой данные для задачи классификации.

В коде последовательно выводятся названия методов выбора признаков и вызывается функция test\_c с соответствующими данными. Например, в первой строке кода выводится строка “VarianceThreshold”, а затем вызывается функция test\_c с аргументами X\_vt\_c и y\_c. Это означает, что функция test\_c будет протестирована на данных, полученных с использованием метода VarianceThreshold.

Аналогичным образом для каждого из методов выбора признаков выводится его название и вызывается функция test\_c с соответствующими данными.

Таким образом, этот код позволяет протестировать функцию test\_c на данных, полученных с использованием различных методов выбора признаков.

19из30

объясни код class PCA\_new():

def \_\_init\_\_(self, n\_components):

self.n\_components = n\_components

self.matrix = np.zeros((1, 1))

def fit(self, X):

X\_cov = np.cov(X)

w, v = np.linalg.eig(X\_cov)

self.matrix = [(i, j) for i, j in zip(w, v)]

self.matrix.sort(key=lambda x: x[0], reverse=True)

self.matrix = np.array([i[1] for i in self.matrix[:]])[:self.n\_components]

return self

def transform(self, X):

basic = X @ (X.T @ self.matrix.T)

return basic

def fit\_transform(self, X):

self.fit(X)

return self.transform(X)

Этот код содержит определение класса PCA\_new, который реализует метод главных компонент (PCA). Класс имеет три метода: \_\_init\_\_, fit и transform.

Метод \_\_init\_\_ принимает один аргумент: n\_components, который указывает, сколько главных компонент должно быть выбрано. Внутри метода создается переменная экземпляра self.n\_components, которая сохраняет значение аргумента n\_components, и переменная экземпляра self.matrix, которая инициализируется нулевой матрицей размера 1x1.

Метод fit принимает один аргумент: X, который представляет собой данные для анализа. Внутри метода вычисляется ковариационная матрица данных X с использованием функции np.cov. Затем вычисляются собственные значения и собственные векторы ковариационной матрицы с использованием функции np.linalg.eig. Собственные значения и собственные векторы сохраняются в переменной экземпляра self.matrix в виде списка кортежей, где первый элемент кортежа - собственное значение, а второй - соответствующий собственный вектор.

Затем список кортежей сортируется в порядке убывания собственных значений и из него выбираются первые self.n\_components собственных векторов. Эти векторы сохраняются в переменной экземпляра self.matrix.

Метод transform принимает один аргумент: X, который представляет собой данные для преобразования. Внутри метода вычисляется произведение данных X и транспонированной матрицы, содержащей выбранные собственные векторы. Результат этого произведения возвращается как результат преобразования.

Метод fit\_transform принимает один аргумент: X, который представляет собой данные для анализа и преобразования. Внутри метода вызывается метод fit с аргументом X, затем вызывается метод transform с аргументом X и возвращается его результат.

Таким образом, этот код реализует класс для выполнения метода главных компонент на данных.

20из30

объясни код X\_pcanew\_c2 = PCA\_new(2).fit\_transform(X\_c[:100])

X\_pcanew\_c3 = PCA\_new(3).fit\_transform(X\_c[:100])

print(X\_pcanew\_c2.shape)

Этот код использует класс PCA\_new, который, вероятно, был определен ранее. Класс PCA\_new реализует метод главных компонент (PCA).

В первой строке кода создается объект PCA\_new с параметром n\_components=2 и вызывается его метод fit\_transform с аргументом X\_c[:100]. Это означает, что будет выполнен метод главных компонент на первых 100 строках данных X\_c и выбраны 2 главные компоненты. Результат преобразования сохраняется в переменной X\_pcanew\_c2.

Во второй строке кода аналогичным образом создается объект PCA\_new с параметром n\_components=3 и вызывается его метод fit\_transform с аргументом X\_c[:100]. Это означает, что будет выполнен метод главных компонент на первых 100 строках данных X\_c и выбраны 3 главные компоненты. Результат преобразования сохраняется в переменной X\_pcanew\_c3.

В третьей строке кода выводится размер массива, сохраненного в переменной X\_pcanew\_c2.

Таким образом, этот код позволяет выполнить метод главных компонент на первых 100 строках данных X\_c и выбрать 2 или 3 главные компоненты.

21из30

объясни код plt.scatter(X\_pcanew\_c2.loc[:,0], X\_pcanew\_c2.loc[:,1], c=classification["Delay"][:100]);

Этот код использует библиотеку matplotlib.pyplot для создания диаграммы рассеяния. В коде вызывается функция scatter с тремя аргументами: X\_pcanew\_c2.loc[:,0], X\_pcanew\_c2.loc[:,1] и c=classification["Delay"][:100].

Первый аргумент X\_pcanew\_c2.loc[:,0] представляет собой значения первого столбца DataFrame X\_pcanew\_c2, которые будут использованы в качестве координат x точек на диаграмме рассеяния.

Второй аргумент X\_pcanew\_c2.loc[:,1] представляет собой значения второго столбца DataFrame X\_pcanew\_c2, которые будут использованы в качестве координат y точек на диаграмме рассеяния.

Третий аргумент c=classification["Delay"][:100] указывает, что цвет точек на диаграмме рассеяния должен соответствовать значениям столбца “Delay” из DataFrame classification, взятых для первых 100 строк.

Таким образом, этот код создает диаграмму рассеяния, на которой отображаются точки с координатами x и y, соответствующими значениям первого и второго столбцов DataFrame X\_pcanew\_c2, и цветом, соответствующим значениям столбца “Delay” из DataFrame classification.

22из30

объясни код fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

ax.scatter(X\_pcanew\_c3.loc[:,0], X\_pcanew\_c3.loc[:,1], X\_pcanew\_c3.loc[:,2], c=classification["Delay"][:100]);

Этот код использует библиотеку matplotlib.pyplot для создания трехмерной диаграммы рассеяния. В первой строке кода вызывается функция figure для создания нового рисунка и результат этого вызова сохраняется в переменной fig.

Во второй строке кода вызывается метод add\_subplot объекта fig с аргументом projection='3d' для добавления нового трехмерного подграфика к рисунку. Результат этого вызова сохраняется в переменной ax.

В третьей строке кода вызывается метод scatter объекта ax с четырьмя аргументами: X\_pcanew\_c3.loc[:,0], X\_pcanew\_c3.loc[:,1], X\_pcanew\_c3.loc[:,2] и c=classification["Delay"][:100].

Первый аргумент X\_pcanew\_c3.loc[:,0] представляет собой значения первого столбца DataFrame X\_pcanew\_c3, которые будут использованы в качестве координат x точек на диаграмме рассеяния.

Второй аргумент X\_pcanew\_c3.loc[:,1] представляет собой значения второго столбца DataFrame X\_pcanew\_c3, которые будут использованы в качестве координат y точек на диаграмме рассеяния.

Третий аргумент X\_pcanew\_c3.loc[:,2] представляет собой значения третьего столбца DataFrame X\_pcanew\_c3, которые будут использованы в качестве координат z точек на диаграмме рассеяния.

Четвертый аргумент c=classification["Delay"][:100] указывает, что цвет точек на диаграмме рассеяния должен соответствовать значениям столбца “Delay” из DataFrame classification, взятых для первых 100 строк.

Таким образом, этот код создает трехмерную диаграмму рассеяния, на которой отображаются точки с координатами x, y и z, соответствующими значениям первого, второго и третьего столбцов DataFrame X\_pcanew\_c3, и цветом, соответствующим значениям столбца “Delay” из DataFrame classification.

23из30

объясни код class LinLasso:

def \_\_init\_\_(self):

self.w = np.array([])

self.y = np.array([])

def fit(self, X, y, learning\_rate = 0.0001, max\_iter = 100, alpha = 1.):

X = np.array(X)

y = np.array(y)

self.w = np.array([0 for i in range(X.shape[1])])

self.w = np.append(self.w, 1)

for t in range(max\_iter):

grad = np.array([])

for i in range(X.shape[1]):

grad = np.append(grad, (X[:, i]\*(y - X.dot(self.w[:-1]) - self.w[-1])).sum() + alpha \* self.w[i])

grad = np.append(grad,(y - X.dot(self.w[:-1]) - self.w[-1]).sum() + alpha \* self.w[-1])

w\_next = self.w - 2 \* grad \* learning\_rate / len(X)

self.w = w\_next

return self

def predict(self, X):

X = np.array(X)

self.y = X.dot(self.w[:-1]) + self.w[-1]

return self.y

l = LinLasso().fit(X\_r\_train[:1000], y\_r\_train[:1000])

Этот код содержит определение класса LinLasso, который реализует линейную регрессию с регуляризацией Lasso. Класс имеет три метода: \_\_init\_\_, fit и predict.

Метод \_\_init\_\_ не принимает аргументов и инициализирует две переменные экземпляра: self.w и self.y, которые представляют собой веса модели и предсказанные значения соответственно.

Метод fit принимает пять аргументов: X, y, learning\_rate, max\_iter и alpha. Аргументы X и y представляют собой данные для обучения модели. Аргумент learning\_rate задает скорость обучения, аргумент max\_iter задает максимальное количество итераций алгоритма, а аргумент alpha задает коэффициент регуляризации.

Внутри метода выполняется инициализация весов модели нулевыми значениями, затем выполняется цикл по количеству итераций. На каждой итерации вычисляется градиент функции потерь с учетом регуляризации Lasso, затем вычисляются новые значения весов модели с использованием градиентного спуска. В конце метода обновленные значения весов сохраняются в переменной экземпляра self.w.

Метод predict принимает один аргумент: X, который представляет собой данные для предсказания. Внутри метода вычисляются предсказанные значения с использованием обученной модели и сохраняются в переменной экземпляра self.y. Затем возвращается массив предсказанных значений.

В последней строке кода создается объект класса LinLasso и вызывается его метод fit с аргументами X\_r\_train[:1000] и y\_r\_train[:1000]. Это означает, что модель будет обучена на первых 1000 строках данных X\_r\_train и соответствующих им значениях целевой переменной из массива y\_r\_train.

Таким образом, этот код реализует класс для выполнения линейной регрессии с регуляризацией Lasso.

24из30

объясни код l.w

l.w обращается к атрибуту w объекта l. В предыдущем коде, который вы предоставили, l был создан как экземпляр класса LinLasso, который реализует линейную регрессию с регуляризацией Lasso. Атрибут w этого класса представляет собой массив весов модели, который инициализируется в методе \_\_init\_\_ и обновляется в методе fit.

Таким образом, l.w возвращает массив весов модели, обученной на данных, которые были переданы в метод fit объекта l.

25из30

объясни код weights = enumerate(l.w)

weights = sorted(weights, key=lambda x: abs(x[1]), reverse=True)

weights

Этот код сортирует веса модели линейной регрессии с регуляризацией Lasso по абсолютному значению. В первой строке кода вызывается функция enumerate с аргументом l.w, который, вероятно, является массивом весов модели, обученной ранее. Функция enumerate возвращает итератор, который генерирует пары индекс-значение для каждого элемента массива l.w. Результат этого вызова сохраняется в переменной weights.

Во второй строке кода вызывается функция sorted с аргументами weights и key=lambda x: abs(x[1]), которые указывают, что нужно отсортировать элементы списка weights по абсолютному значению второго элемента каждой пары. Также передается аргумент reverse=True, который указывает, что сортировка должна быть выполнена в порядке убывания. Результат этого вызова сохраняется в переменной weights.

В третьей строке кода выводится содержимое переменной weights.

Таким образом, этот код сортирует веса модели линейной регрессии с регуляризацией Lasso по абсолютному значению и выводит отсортированный список.