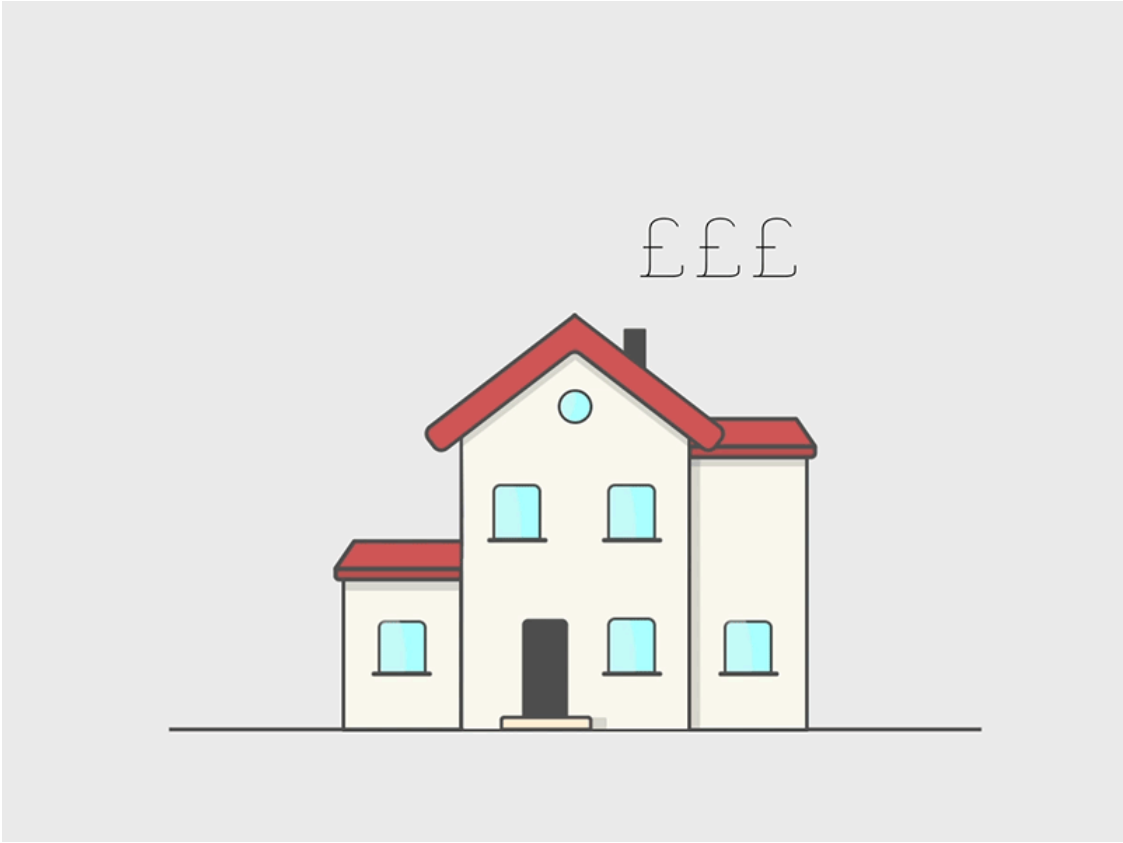
Regresja liniowa i logistyczna

Wstęp

Celem tego laboratorium będzie stworzenie modelu uczenia maszynowego do estymacji cen nieruchomości na podstawie danych o jej położeniu, ilości sypialń, roku budowy, typie budynku oraz wielu innych parametrów.



W trakcie realizacji tego labratorium zapoznamy się z następującymi zagadnieniami:

- przygotowaniem danych:
  - ładowaniem danych,
  - typami danych,
  - czyszczeniem danych,
  - rozkładami danych,
  - obsługą wartości brakujących,
  - zmiennymi kategorycznymi uporządkowanymi i nieuporządkowanymi,
  - skalowaniem wartości,
  - API biblioteki Scikit-Learn dla transformacji danych;
- regresją liniową, w szczególności z:
  - podziałem zbioru na część treningową i testową,
  - oceną jakości modelu,
  - walidacją skrośną,
  - wyszukiwaniem hiperparametrów,
  - problemem przeuczenia, niedouczenia,
  - regularyzacją L1 i L2,
  - regresją wielomianową;
- regresją logistyczną, w szczególności z:
  - różnymi rodzajami błędów klasyfikacji,
  - metrykami oceniającymi jakość klasyfikatorów.

Na pierwszych zajęciach możesz korzystać ze środowiska Google Colab i zdalnego środowiska obliczeniowego. Jeżeli interesuje Cię skonfigurowanie Pythona na własnym komputerze, to niezbędne informacje są podane w sekcji "Konfiguracja własnego komputera".

Uwaga: niektóre zadania zamiast kodu wymagają podania pisemnej odpowiedzi w miejscu oznaczonym `// skomentuj tutaj`.

Wykorzystywane biblioteki

Na zajęciach korzystać będziesz z kilku popularnych bibliotek Pythona, które umożliwiają klasyfikację danych, ich wizualizację czy preprocessing. Są to:

- numpy - bibliotek do wykonywania obliczeń macierzowych. Pozwala na efektywne przeprowadzanie obliczeń naukowych. Dobrze współgra z biblioteką pandas.
- pandas - narzędzie do analizy danych tabelarycznych, ich strukturyzowania oraz manipulacji na nich.
- sklearn - narzędzie do tworzenia modeli klasyfikacji, regresji, clusteringu itp. Biblioteka ta jest dość rozbudowana i pozwala także na mapowanie danych czy redukcję wymiarów. Więcej informacji znajdziesz w podanym linku.
- missingno - narzędzie do wizualizacji kompletności danych (brakujących wartości).
- seaborn - kompleksowe narzędzie do wizualizacji danych jako takich. Pozwala na stworzenie bardzo szerokiej gamy wykresów w zależności od potrzeb.

Zostały tutaj pominięte pewne standardowe biblioteki jak np. `os` czy `matplotlib`.

Wykorzystanie Google Colab

Korzystanie Google Colab nie jest wymagane. W niektórych laboratorich może być jednak przydatny dostęp do środowiska wyposażonego w kartę GPU.

Open in Colab

Jeżeli pracujesz na Google Colab, zacznij od przeniesienia dwóch plików CSV, które zostały dołączone do laboratorium (`ames_data.csv` oraz `bank_marketing_data.csv`), do folderu `/content`. Nie musisz ich umieszczać w `/content/sample_data` - ważne, aby znalazły się w `/content`. Jeżeli pracujesz lokalnie, to wystarczy, że pliki te będą obok tego notebooka.

Konfiguracja własnego komputera

Jeżeli korzystasz z własnego komputera, to musisz zainstalować trochę więcej bibliotek (Google Colab ma je już zainstalowane). Najlepiej używać Pythona 3.9 lub nowszej wersji. Laboratorium było testowane z wersją 3.9.

Anaconda

Jeżeli korzystasz z Anacondy (możesz uruchomić w terminalu):

In [ ]:
```
# !conda install -c conda-forge --yes numpy pandas scikit-learn matplotlib missingno
```

venv

Jeżeli używasz zwykłego venv'a (**zdecydowanie niezalecane, szczególnie na Windowsie**):

In [ ]:
```
# !pip install --yes numpy pandas scikit-learn matplotlib missingno
```

W przypadku własnego komputera, jeżeli instalowałeś z terminala, pamiętaj, aby zarejestrować aktualne środowisko wirtualne jako kernel (środowisko uruchomieniowe) dla Jupyter Notebooka. Wybierz go jako używany kernel w menu na górze notebooka (nazwa jak w komendzie poniżej).

In [ ]:
```
# !ipython kernel install --user --name "PSI_3.9"
```

Zbiór danych do regresji

Wykorzystamy zbiór danych Ames housing, w którym zadaniem jest przewidywanie wartości domu na podstawie cech budynku, działki, lokalizacji itp. Jest to więc przewidywanie wartości ciągłej, czyli regresja. Zbiór ten zawiera zmienne numeryczne (floaty i inty), kategoryczne nieuporządkowane (*categorical nominal*) oraz kategoryczne uporządkowane (*categorical ordinal*), więc będzie wymagał wstępnego przetworzenia tak jak większość prawdziwych danych w uczeniu maszynowym.

Inne znane, ale gorsze jakościowo zbiory tego typu, to na przykład:

- Boston housing - rasistowski, z tego powodu usunięty np. ze Scikit-learn (wyjaśnienie, dyskusja, badanie)
- California housing - zbyt prosty (tylko kilka zmiennych numerycznych), użyty np. w książce "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" A. Geron (opis)

Autor zbioru to Dean De Cock, a zbiór został opisany oryginalnie w tym artykule.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Ładowanie danych tabelarycznych

Pliki `ames_data.csv` oraz `bank_marketing_data.csv` to dwa zbiory danych, niezależne od siebie. Pierwszy jest wykorzystywany w pierwszej części laboratorium (regresji liniowej), natomiast drugi przyda się przy regresji logistycznej (klasyfikacji). Jego celem jest przewidywanie wartości domu.

Wczytajmy dane `ames_data.csv` do zmiennej `df` (takiej nazwy często się używa, żeby oznaczyć obiekt `DataFrame` - zaawansowanej tablicy, dostarczonej nam przez bibliotekę `pandas`).

In [ ]:
```
df = pd.read_csv("ames_data.csv")

# remove dots from names to match data_description.txt
df.columns = [col.replace(".", "") for col in df.columns]
```

Zobaczmy jakie dane znajdują się w naszej tabeli. Wykorzystajmy do tego metodę `info()`.

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 82 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Order           2930 non-null   int64
 1   PID             2930 non-null   int64
 2   MSSubClass      2930 non-null   int64
 3   MSZoning        2930 non-null   object
 4   LotFrontage     2440 non-null   float64
 5   LotArea         2930 non-null   int64
 6   Street          2930 non-null   object
 7   Alley           198 non-null    object
 8   LotShape        2930 non-null   object
 9   LandContour     2930 non-null   object
 10  Utilities       2930 non-null   object
 11  LotConfig       2930 non-null   object
 12  LandSlope       2930 non-null   object
 13  Neighborhood    2930 non-null   object
 14  Condition1      2930 non-null   object
 15  Condition2      2930 non-null   object
 16  BldgType        2930 non-null   object
 17  HouseStyle      2930 non-null   object
 18  OverallQual     2930 non-null   int64
 19  OverallCond     2930 non-null   int64
 20  YearBuilt       2930 non-null   int64
 21  YearRemodAdd    2930 non-null   int64
 22  RoofStyle       2930 non-null   object
 23  RoofMatl        2930 non-null   object
 24  Exterior1st     2930 non-null   object
 25  Exterior2nd     2930 non-null   object
 26  MasVnrType      1155 non-null   object
 27  MasVnrArea      2907 non-null   float64
 28  ExterQual       2930 non-null   object
 29  ExterCond       2930 non-null   object
 30  Foundation      2930 non-null   object
 31  BsmtQual        2850 non-null   object
 32  BsmtCond        2850 non-null   object
 33  BsmtExposure    2847 non-null   object
 34  BsmtFinType1    2850 non-null   object
 35  BsmtFinSF1      2929 non-null   float64
 36  BsmtFinType2    2849 non-null   object
 37  BsmtFinSF2      2929 non-null   float64
 38  BsmtUnfSF       2929 non-null   float64
 39  TotalBsmtSF     2929 non-null   float64
 40  Heating         2930 non-null   object
 41  HeatingQC       2930 non-null   object
 42  CentralAir      2930 non-null   object
 43  Electrical      2929 non-null   object
 44  X1stFlrSF       2930 non-null   int64
 45  X2ndFlrSF       2930 non-null   int64
 46  LowQualFinSF    2930 non-null   int64
 47  GrLivArea       2930 non-null   int64
 48  BsmtFullBath    2928 non-null   float64
 49  BsmtHalfBath    2928 non-null   float64
 50  FullBath        2930 non-null   int64
 51  HalfBath        2930 non-null   int64
 52  BedroomAbvGr    2930 non-null   int64
 53  KitchenAbvGr    2930 non-null   int64
 54  KitchenQual     2930 non-null   object
 55  TotRmsAbvGrd    2930 non-null   int64
 56  Functional      2930 non-null   object
 57  Fireplaces      2930 non-null   int64
 58  FireplaceQu     1508 non-null   object
 59  GarageType      2773 non-null   object
 60  GarageYrBlt     2771 non-null   float64
 61  GarageFinish    2771 non-null   object
 62  GarageCars      2929 non-null   float64
 63  GarageArea      2929 non-null   float64
 64  GarageQual      2771 non-null   object
 65  GarageCond      2771 non-null   object
 66  PavedDrive      2930 non-null   object
 67  WoodDeckSF      2930 non-null   int64
 68  OpenPorchSF     2930 non-null   int64
 69  EnclosedPorch   2930 non-null   int64
 70  X3SsnPorch      2930 non-null   int64
 71  ScreenPorch     2930 non-null   int64
 72  PoolArea        2930 non-null   int64
 73  PoolQC          13 non-null     object
 74  Fence           572 non-null    object
```

```
 75  MiscFeature    106 non-null    object
 76  MiscVal        2930 non-null   int64
 77  MoSold         2930 non-null   int64
 78  YrSold         2930 non-null   int64
 79  SaleType       2930 non-null   object
 80  SaleCondition  2930 non-null   object
 81  SalePrice      2930 non-null   int64
dtypes: float64(11), int64(28), object(43)
memory usage: 1.8+ MB
```

Mamy naprawdę dużo cech! Ich szczegółowy opis znajdziesz w dołączonym do laboratorium pliku `ames_description.txt`.

Wstępna analiza danych

Zawsze, zanim zaczniesz robić jakąkolwiek predykcję czy analizę danych, dobrze jest zapoznać się z nimi, z ich kodowaniem i znaczeniem. Kolejnym istotnym aspektem jest typ danych. Nie każdy klasyfikator nadaje się do każdego typu.

Wyświetlmy teraz kilka przykładowych rekordów z początku pliku, korzystając z metody `head()`.

In [ ]:
```
df.head()
```

Out[ ]:

	Order	PID	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	...	PoolArea	Po
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...	0	
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...	0	
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	...	0	
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...	0	
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...	0	

5 rows × 82 columns

Jeżeli potrzebujesz szybko stwierdzić, ile dane zawierają rekordów i kolumn, pomocna jest opcja `shape`:

In [ ]:
```
df.shape
```

Out[ ]:
```
(2930, 82)
```

Eksploracja danych, czyszczenie danych i inżynieria cech

Usunięcie niepotrzebnych kolumn

Niektóre kolumny są **nieinformatywne (uninformative)**, czyli nie niosą żadnej informacji dla zadania, czyli przewidywania wartości domu. Są pewnym rodzajem metadanych. Przykładowo mamy tutaj kolumny **Order** oraz **PID**.

Order jest po prostu numerem rekordu w zbiorze danych, moglibyśmy przetasować cały zbiór i to nie powinno w żaden sposób wpłynąć na cokolwiek, a więc możemy spokojnie tę kolumnę usunąć.

Formalnie czynimy założenie, że rekordy w naszych danych (próbki / wiersze, poszczególne domy w przypadku tego zbioru) są **niezależne i równomiernie rozłożone** (ang. **independent and identically distributed - i.i.d.**). Innymi słowy, kolejność w danych nie ma znaczenia, bo zbieraliśmy dane taką samą metodą i w identycznych warunkach. Jest to bardzo typowe w ML.

PID jest po prostu numerem identyfikacyjnym danej nieruchomości w systemie informatycznym, a więc też możemy to usunąć.

In [ ]:
```
df = df.drop(["Order", "PID"], axis="columns")
```

Usunięcie słabo reprezentowanych dzielnic

Dzielnice *GrnHill* oraz *Landmrk* obejmują w sumie zaledwie 3 domy.

In [ ]:
```
df = df.loc[~df["Neighborhood"].isin(["GrnHill", "Landmrk"]), :]
```

Usunięcie obserwacji odstających (outliers)

Usuniemy budynki, które mają powyżej 4000 stóp kwadratowych (ok. 370 metrów kwadratowych) powierzchni. Możemy zobaczyć je na wykresie poniżej.

In [ ]:
```
plt.scatter(df["GrLivArea"], df["SalePrice"])
plt.title("House area vs price")
plt.xlabel("GrLivArea")
```

```
plt.ylabel("SalePrice")
plt.show()
```

Jak widać na wykresie, jest dosłownie kilka domów o tej powierzhcni. Takie skrajne przypadki raczej nas nie interesują - a na pewno stanowią problem dla tak prostego modelu jak regresja logistyczna. Nie chcemy też, żeby nasz model uczył się takich anomalii, więc lepiej je usunąć.

Tutaj robimy to ręcznie, ale istnieją też algorytmy do detekcji i usuwania obserwacji odstających.

Zadanie 1 (0.25 punktu)

Usuń rekordy nieruchomości o powierzchni (**GrLivArea**) ponad (ostra nierówność) 4 tys. stóp kwadratowych.

Podpowiedź: w Pandas korzysta się z `.loc[]` do filtrowania wierszy i kolumn. Pierwszy indeks oznacza, które wiersze zostawić, a drugi indeks, które kolumny wybrać. Jeżeli chcemy zostawić wszystko (np. nie usuwać żadnych kolumn), to zadziała standardowy Pythonowy `:`, jak przy indeksowaniu list.

In [ ]:
```
# remove outliers
df = df.loc[~df["GrLivArea"].gt(4000), :]
```

Zobaczmy jak teraz wygląda ten sam wykres.

In [ ]:
```
plt.scatter(df["GrLivArea"], df["SalePrice"])
plt.title("House area vs price, outliers removed")
plt.xlabel("GrLivArea")
plt.ylabel("SalePrice")
plt.show()
```

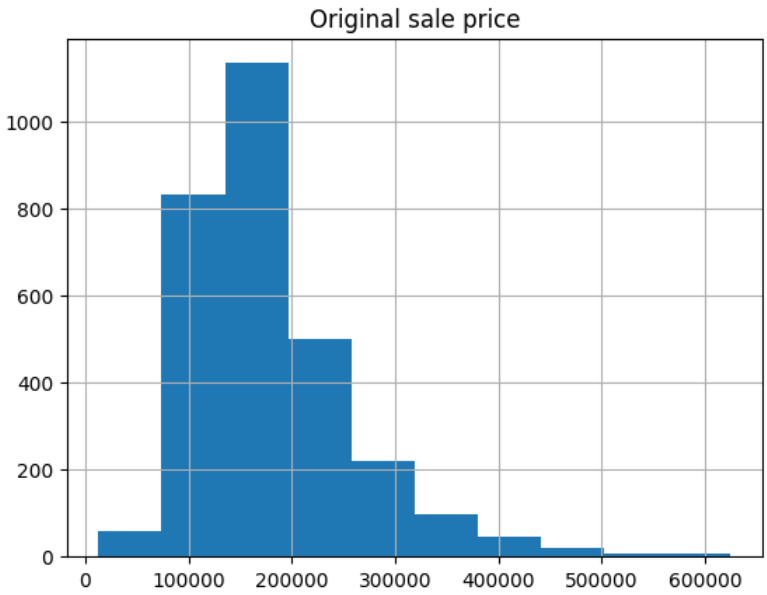
Transformacja logarytmiczna zmiennej zależnej

Zawsze warto też przyjrzeć się rozkładowi zmiennej docelowej, żeby poznać jej typ i skalę. Jak widać poniżej, rozkład jest dość skośny, co ma sens - mało jest bardzo drogich domów.

In [ ]:
```
df["SalePrice"].describe()
```

Out[ ]:
```
count      2922.000000
mean     180358.266940
std       78536.952287
min       12789.000000
25%      129425.000000
50%      160000.000000
75%      213430.000000
max      625000.000000
Name: SalePrice, dtype: float64
```

In [ ]:
```
df["SalePrice"].hist()
plt.title("Original sale price")
plt.show()
```



Rozkład normalny jest zwykle korzystniejszy dla tworzenia modeli, bo daje sensowną "wartość środkową" do przewidywania, a także penalizuje tak samo błędy niezależnie od ich znaku (zaniżona i zawyżona predykcja). Dokonamy dlatego **transformacji logarytmicznej (log transform)**, czyli zlogarytmujemy zmienną docelową (zależną). Dla stabilności numerycznej używa się zwykle `np.log1p`, a nie `np.log` (tutaj wyjaśnienie).

Dodatkowa korzyść z takiej transformacji jest taka, że regresja liniowa przewiduje dowolne wartości rzeczywiste. Po przekształceniu logarytmicznym jest to całkowicie ok, natomiast w oryginalnej przestrzeni trzeba by wymusić przewidywanie tylko wartości pozytywnych (negatywne ceny są bez sensu). Da się to zrobić, ale zwiększa to koszt obliczeniowy. Operowanie na tzw. log-price jest bardzo częste w finansach.

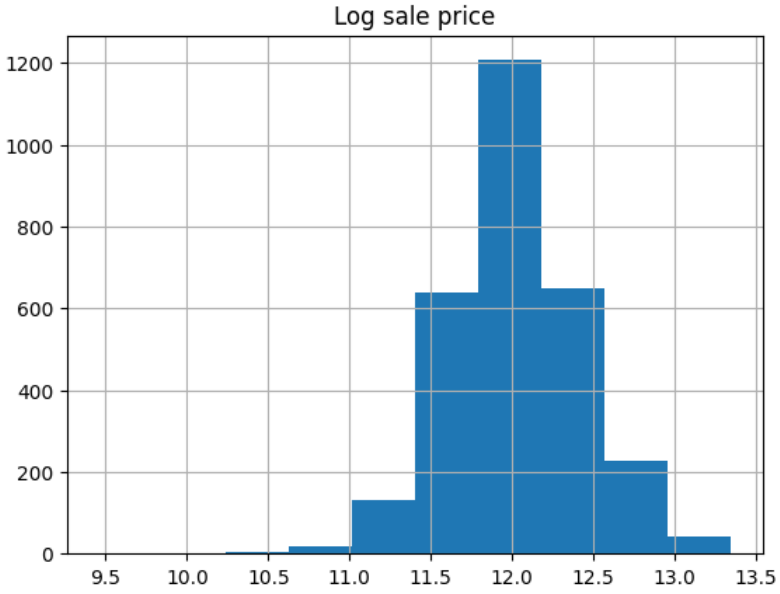
Zadanie 2 (0.25 punktu)

Przekształć zmienną **SalePrice** za pomocą funkcji logarytmicznej `np.log1p`.

In [ ]:
```
# apply log transform
df['SalePrice'] = df['SalePrice'].transform(lambda x : np.log1p(x))
```

Sprawdźmy teraz jak rozkład **SalePrice** wygląda po transformacji:

In [ ]:
```
pd.Series(df["SalePrice"]).hist()
plt.title("Log sale price")
plt.show()
```

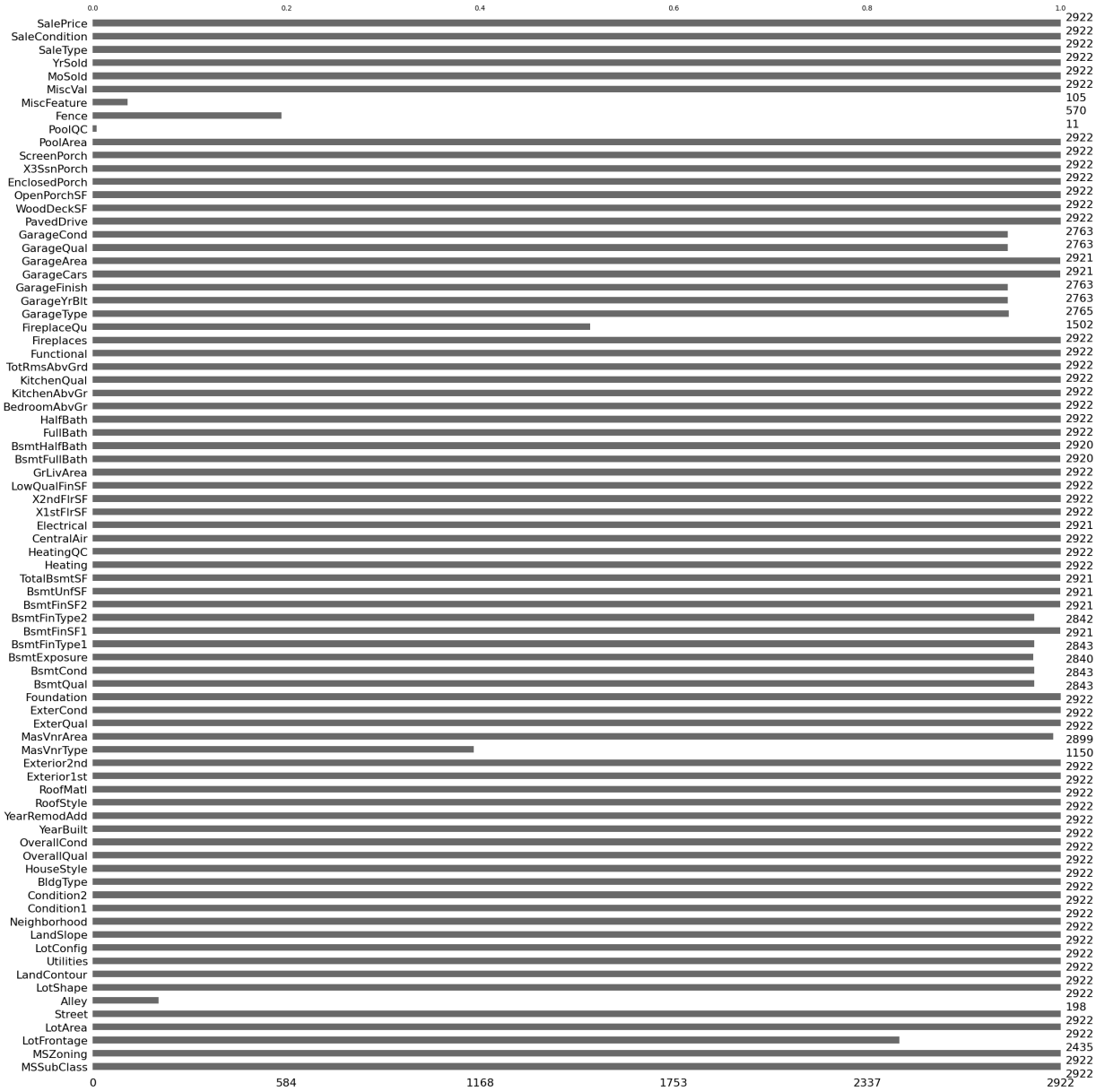


Uzupełnianie wartości brakujących

Sprawdźmy też wartości brakujące. Są zmienne, które mają poniżej 10% wartości - takie zmienne dla modeli regresji liniowej są po prostu bezużyteczne, ponieważ brakujących wartości nie można wprost zamodelować. Znacząca liczba cech ma jednak co najmniej 10% braków. Z nich będziemy jednak starali się zrobić użytek.

In [ ]:
```
import missingno as msno

msno.bar(df)
```

Out[ ]:
```
<Axes: >
```



W ramach dalszego czyszczenia danych automatycznie uzupełnimy wartości brakujące. Trzeba tu jednak wziąć pod uwagę:

- zmienne kategoryczne - nie można w nich dokonać zastąpienia wartości brakującej średnią, medianą itp.
- wiele brakujących wartości - estymacja modą czy medianą byłaby niedokładna,
- możliwość wykorzystania wiedzy o innych zmiennych na podstawie opisu cech.

Można więc zastosować odpowiednią wiedzę i przyjąć wartości domyślne. Przykładowo, brak informacji o powierzchni piwnicy możemy uznać po prostu za brak piwnicy i wpisać tam odpowiednią wartość. W przypadku niektórych zmiennych może doprowadzić to do stworzenia nowej wartości, która implicite będzie reprezentować wartość brakującą.

Znaczna część poniższej analizy została zainspirowana tym notebookiem na Kaggle.

In [ ]:
```
def replace_na(df: pd.DataFrame, col: str, value) -> None:
    df.loc[:, col] = df.loc[:, col].fillna(value)
```

In [ ]:
```
# Alley : data description says NA means "no alley access"
replace_na(df, "Alley", value="None")

# BedroomAbvGr : NA most likely means 0
replace_na(df, "BedroomAbvGr", value=0)

# BsmtQual etc : data description says NA for basement features is "no basement"
replace_na(df, "BsmtQual", value="No")
replace_na(df, "BsmtCond", value="No")
replace_na(df, "BsmtExposure", value="No")
replace_na(df, "BsmtFinType1", value="No")
replace_na(df, "BsmtFinType2", value="No")
replace_na(df, "BsmtFullBath", value=0)
replace_na(df, "BsmtHalfBath", value=0)
replace_na(df, "BsmtUnfSF", value=0)
```

```
# Condition : NA most likely means Normal
replace_na(df, "Condition1", value="Norm")
replace_na(df, "Condition2", value="Norm")

# External stuff : NA most likely means average
replace_na(df, "ExterCond", value="TA")
replace_na(df, "ExterQual", value="TA")

# Fence : data description says NA means "no fence"
replace_na(df, "Fence", value="No")

# Functional : data description says NA means typical
replace_na(df, "Functional", value="Typ")

# GarageType etc : data description says NA for garage features is "no garage"
replace_na(df, "GarageType", value="No")
replace_na(df, "GarageFinish", value="No")
replace_na(df, "GarageQual", value="No")
replace_na(df, "GarageCond", value="No")
replace_na(df, "GarageArea", value=0)
replace_na(df, "GarageCars", value=0)

# HalfBath : NA most likely means no half baths above grade
replace_na(df, "HalfBath", value=0)

# HeatingQC : NA most likely means typical
replace_na(df, "HeatingQC", value="Ta")

# KitchenAbvGr : NA most likely means 0
replace_na(df, "KitchenAbvGr", value=0)

# KitchenQual : NA most likely means typical
replace_na(df, "KitchenQual", value="TA")

# LotFrontage : NA most likely means no lot frontage
replace_na(df, "LotFrontage", value=0)

# LotShape : NA most likely means regular
replace_na(df, "LotShape", value="Reg")

# MasVnrType : NA most likely means no veneer
replace_na(df, "MasVnrType", value="None")
replace_na(df, "MasVnrArea", value=0)

# MiscFeature : data description says NA means "no misc feature"
replace_na(df, "MiscFeature", value="No")
replace_na(df, "MiscVal", value=0)

# OpenPorchSF : NA most likely means no open porch
replace_na(df, "OpenPorchSF", value=0)

# PavedDrive : NA most likely means not paved
replace_na(df, "PavedDrive", value="N")

# PoolQC : data description says NA means "no pool"
replace_na(df, "PoolQC", value="No")
replace_na(df, "PoolArea", value=0)

# SaleCondition : NA most likely means normal sale
replace_na(df, "SaleCondition", value="Normal")

# ScreenPorch : NA most likely means no screen porch
replace_na(df, "ScreenPorch", value=0)

# TotRmsAbvGrd : NA most likely means 0
replace_na(df, "TotRmsAbvGrd", value=0)

# Utilities : NA most likely means all public utilities
replace_na(df, "Utilities", value="AllPub")

# WoodDeckSF : NA most likely means no wood deck
replace_na(df, "WoodDeckSF", value=0)
```

W przypadku wykonywania tego typu zmian - o ile istnieje taka możliwość - warto rozważyć różne interpretacje brakujących wartości. Może okazać się, że przyjęte przez nas założenia są błędne i prowadzą do pogorszenia działania modelu. Dlatego warto porównać jakoś predykcji z danymi uzupełnionymi oraz z danymi, w których kolumna z brakującymi wartościami jest po prostu usuwana.

Zadanie 3 (0.5 punktu)

Z pomocą dokumentacji zmiennych w pliku `data_description.txt` zdecyduj, jakie wartości domyślne przypisać zmiennym:

- `CentralAir`
- `EnclosedPorch`
- `FireplaceQu` oraz `Fireplaces`
- `SaleCondition`

W praktyce niestety zwykle nie jest tak łatwo, że mamy dokumentację i ten krok zajmuje kilka godzin (lub dni) konsultacji z różnymi osobami w firmie :) Czasami w ogóle nie da się ustalić jaka wartość byłaby sensowna, ponieważ nie mamy żadnego dostępu do osób odpowiedzialnych za przygotowanie wykorzystywanego zbioru danych.

In [ ]:
```
# CentralAir : 90% of households in the US possess Central Air Conditioning whereas only 10% in the EU
replace_na(df, "CentralAir", value="N")

# EnclosedPorch :  Endclosed Porch is not popular
replace_na(df, "EnclosedPorch", value=0)

# Fireplaces : aproximately there are 1.3 fireplace per house
replace_na(df, "Fireplaces", value=1)
replace_na(df, "FireplaceQu", value="TA")

# SaleCondition
replace_na(df, "SaleCondition", value="Normal")
```

Dane kategoryczne

Jak już zdążyliśmy zauważyć, istnieją dwa główne rodzaje danych: numeryczne (*numerical data*) oraz kategoryczne (*categorical data*). Ten podział jest bardzo istotny. Dane numeryczne to żadna niespodzianka, po prostu mają swoją wartość, jak np. **GrLivArea**, czyli powierzchnia budynku/apartamentów. Dane kategoryczne to takie, którym w większości przypadków nie można przyporządkować wartości liczbowej (wyjątkiem są dane kategoryczne uporządkowane - *categorical ordinal*).

Wyobraź sobie zmienną reprezentującą kolory o wartościach "red", "green" i "blue. Jeżeli zakodowałbyś je np. jako $red = 0$, $green = 1$, $blue = 2$, to stwierdzasz tym samym, że w pewnym sensie $red < green < blue$. Raczej nie ma powodu, żeby tak sądzić. Jest to zmienna, która ma skończoną liczbę wartości, ale są one nieuporządkowane. Taki typ to zmienne *categorical nominal*.

Szczególnym przypadkiem są zmienne binarne (*boolean*). Jest to u nas kolumna **CentralAir** (Central Air Conditioning). Z opisu w pliku `ames_description.txt` wiemy, że przyjmuje ona dokładnie dwie wartości kategoryczne: *No* oraz *Yes*. W takiej sytuacji wolno zakodować te wartości numerycznie jako 0 i 1. Stwierdzasz tym samym, że klimatyzacja albo jest, albo jej nie ma.

Sytuacją podobną, chociaż mniej oczywistą, może być zmienna **Street**, opisująca typ drogi wiodącej do nieruchomości. Jeśli znowu spojrzymy do opisu danych, to można zauważyć, że ta zmienna może przyjmować tylko dwie różne wartości - *Grvl* i *Pave*. I tu też możemy sobie pozwolić na zakodowanie tych wartości jako 0 i 1. Stwierdzamy wtedy, że droga jest *utwardzona* (Pave) dla wartości 1. Oczywiście równie dobrze można by zakodować to odwrotnie i stwierdzić, że droga jest *nieutwardzona* (Grvl) gdy wartość wynosi 1.

W Pandas typy numeryczne są oparte o NumPy (np. `np.int64`), a zmienne kategoryczne, napisy itp. są typu `object` (typ `Categorical` istnieje od pewnego czasu, ale nie jest jeszcze zbyt dobrze wspierany).

Zmienne **MSSubClass** oraz **MoSold** są kategoryczne (tak wynika z informacji zawartej w pliku `ames_description.txt`), a są w naszych danych wprost liczbami. Przekształćmy je zatem do poprawnego typu.

In [ ]:
```
df = df.replace(
    {
        "MSSubClass": {
            20: "SC20",
            30: "SC30",
            40: "SC40",
            45: "SC45",
            50: "SC50",
            60: "SC60",
            70: "SC70",
            75: "SC75",
            80: "SC80",
            85: "SC85",
            90: "SC90",
            120: "SC120",
            150: "SC150",
            160: "SC160",
            180: "SC180",
            190: "SC190",
        },
        "MoSold": {
            1: "Jan",
            2: "Feb",
            3: "Mar",
```

```
            4: "Apr",
            5: "May",
            6: "Jun",
            7: "Jul",
            8: "Aug",
            9: "Sep",
            10: "Oct",
            11: "Nov",
            12: "Dec",
        },
    }
)
```

Oprócz tego zakodujemy zmienne kategoryczne uporządkowane (*categorical ordinal*) z tekstowych na kolejne liczby całkowite.

Przykładowo zmienna **BsmtCond**, oceniająca stan piwnicy, ma następujące możliwe wartości:

- *NA* (No) Basement
- *Po* (Poor) - Severe cracking, settling, or wetness
- *Fa* (Fair) - dampness or some cracking or settling
- *TA* (Typical) - slight dampness allowed
- *Gd* (Good)
- *Ex* (Excellent)

Do następujących wartości możemy dopasować pewną skalę punktową, bo są one naturalnie uporządkowane.

In [ ]:
```
df = df.replace(
    {
        "Alley": {"None": 0, "Grvl": 1, "Pave": 2},
        "BsmtCond": {"No": 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "BsmtExposure": {"No": 0, "Mn": 1, "Av": 2, "Gd": 3},
        "BsmtFinType1": {
            "No": 0,
            "Unf": 1,
            "LwQ": 2,
            "Rec": 3,
            "BLQ": 4,
            "ALQ": 5,
            "GLQ": 6,
        },
        "BsmtFinType2": {
            "No": 0,
            "Unf": 1,
            "LwQ": 2,
            "Rec": 3,
            "BLQ": 4,
            "ALQ": 5,
            "GLQ": 6,
        },
        "BsmtQual": {"No": 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "ExterCond": {"Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "ExterQual": {"Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "FireplaceQu": {"No": 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "Functional": {
            "Sal": 1,
            "Sev": 2,
            "Maj2": 3,
            "Maj1": 4,
            "Mod": 5,
            "Min2": 6,
            "Min1": 7,
            "Typ": 8,
        },
        "GarageCond": {"No": 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "GarageQual": {"No": 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "HeatingQC": {"Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "KitchenQual": {"Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5},
        "LandSlope": {"Sev": 1, "Mod": 2, "Gtl": 3},
        "LotShape": {"IR3": 1, "IR2": 2, "IR1": 3, "Reg": 4},
        "PavedDrive": {"N": 0, "P": 1, "Y": 2},
        "PoolQC": {"No": 0, "Fa": 1, "TA": 2, "Gd": 3, "Ex": 4},
        "Street": {"Grvl": 0, "Pave": 1},
        "Utilities": {"ELO": 1, "NoSeWa": 2, "NoSewr": 3, "AllPub": 4},
    }
)
```

Przygotowanie danych do uczenia

Nasz zbiór podzielimy na dwa podzbiory: treningowy (70%) i testowy (30%). Zbiór treningowy pozwoli nam utworzyć model regresji liniowej, natomiast testowy - oszacować jego jakość.

Pamiętaj, że wyniki uzyskiwane przez model na danych treningowych nie odzwierciedlają tego, jak będzie on sobie radził na danych, których nie ma w zbiorze uczącym. Aby uzyskać taką informację, konieczne jest sprawdzenie, jak model radzi sobie na danych testowych. Daje nam to oszacowanie, jak dobrze model **generalizuje się** dla nowych danych.

Wydzielimy sobie również zbiory kolumn z danymi numerycznymi i kategorycznymi, co później ułatwi nam odwoływanie się do nich.

Funkcja `train_test_split` z biblioteki Scikit-Learn przyjmuje osobno macierze dla cech (*features*) i etykiet (*labels*), dlatego wyodrębniamy sobie z naszej tablicy kolumnę **SalePrice**, która zawiera ceny nieruchomości.

Ciekawostka

Można zauważyć, że zmienna `y` jest małą literą, natomiast `X_train` czy `X_test` są z dużej. Są to konwencje pochodzące z matematyki:

- wektor w matematyce często oznaczamy małą pogrubioną literą ($\mathbf{y}$) - w programowaniu natomiast oznaczamy po prostu małą literą - `y`
- macierz w matematyce oznaczamy dużą pogrubioną literą ($\mathbf{X}$) - w programowaniu po prostu dużą literą - `X`

Zbiór etykiet to w naszym przypadku wektor cen, więc zapisujemy `y` małą literą. Z drugiej strony `X` zawiera kolumny z cechami opisującymi poszczególne rekordy, a więc jest to macierz.

Uwaga: w eksperymentach ustalamy na sztywno wartość parametru `random_state`. Doczytaj, dlaczego wykorzystywany jest ten parametr i co się dzieje, gdy jest on równy stałej wartości jak zero.

In [ ]:
```
from sklearn.model_selection import train_test_split

y = df.pop("SalePrice")

categorical_features = df.select_dtypes(include="object").columns
numerical_features = df.select_dtypes(exclude="object").columns

X_train, X_test, y_train, y_test = train_test_split(
    df, y, test_size=0.3, random_state=0
)
```

In [ ]:
```
df_ames = df.copy()
y_ames = y.copy()
```

Teraz trzeba dokonać transformacji naszych danych:

- zmienne kategoryczne nieuporządkowane trzeba przetworzyć tak, aby nasz algorym był w stanie je obsłużyć, czyli je zakodować za pomocą **one-hot encoding**,
- zmienne numeryczne dalej mogą mieć wartości brakujące, więc trzeba je uzupełnić, inaczej **imputować (impute)**,
- zmienne numeryczne trzeba przeskalować do zakresu wartości $[0, 1]$ czyli je **znormalizować (normalization)** przez zastosowanie **min-max scaling**.

Kodowanie one-hot encoding

Powyżej omawialiśmy zmienne kategoryczne. Typ *categorical ordinal* można zakodować kolejnymi liczbami całkowitymi, co jest oczywiście proste. Co jednak ze zmiennymi bez kolejności, typu *categorical nominal*? Trzeba je dalej przekształcić na liczby (żeby model był w stanie je przetworzyć), ale tak, aby nie nadać im implicite kolejności.

Spójrzmy na kolumnę **Neighborhood**, oznaczającą poszczególne dzielnice. Dom znajduje się tylko w jednej dzielnicy, a w pozostałych go nie ma. Idea kodowania **one-hot encoding** polega na stworzeniu tylu zmiennych, ile jest możliwych wartości, a następnie w każdym wierszu przypisanie wartości 1 w tej kolumnie, z której była oryginalnie zmienna.

Przykładowo, jeżeli mielibyśmy 3 wartości `["A", "B", "C"]`, to powstają z nich 3 cechy (kolumny macierzy `X`) `[col_A, col_B, col_C]`. Wiersz z pierwotną wartością `"B"` będzie miał wartości tych cech `[0, 1, 0]`. W przypadku naszej zmiennej **Neighborhood** pojawią się osobne zmienne **Old Town**, **NoRidge**, **Gilbert** itd., a dla każdego wiersza dokładnie jedna z nich będzie miała wartość 1.

Dla zainteresowanych

Jeżeli mamy dużo możliwych wartości, czyli zmienną o dużej **kardynalności (cardinality)**, to kolumn powstanie bardzo dużo. Do tego są **rzadkie (sparse)**, więc tracimy dużo pamięci na przechowywanie zer. Istnieją inne kodowania, które zajmują mniej miejsca, a

implementuje je biblioteka Category Encoders.

Imputacja brakujących wartości numerycznych

Wcześniej już napotkaliśmy wartości brakujące i postaraliśmy się uzupełnić je jak najlepiej potrafiliśmy, używając dokumentacji naszego zbioru. Nie gwarantuje to jednak usunięcia wszystkich braków. Nie zawsze w praktyce da się też tak łatwo znaleźć wartości do uzupełnienia. W przypadku zwykłych cech numerycznych możemy zastosować jedną z kilku bardzo popularnych strategii radzenia sobie z wartościami brakującymi:

1. Usunąć kolumnę, która zawiera brakujące wartości.
2. Usunąć wiersze, w których brakuje wartości.
3. Zastąpić brakujące wartości innymi, np. średnią z kolumny, medianą albo wartością stałą.
4. Przewidzieć brakujące wartości wykorzystując odpowiedni model uczenia maszynowego.

Podejście 4 jest często zbyt czasochłonne. Opcje 1 i 2 prowadzą do utraty danych. My wypróbujemy sposób nr 3.

Nie znaczy to jednak, że usunięcie wierszy czy kolumny jest zawsze złym podejściem. Usunięcie kolumny jest uzasadnione, jeśli ma ona naprawdę dużo wartości brakujących. W takich wypadkach ciężko z niej wyciągnąć jakąkolwiek sensowną informację. Usunięcie wierszy może być uzasadnione w przypadku, gdy mamy dużo rekordów i tylko niewielka część z nich posiada wartości brakujące (usunięcie kilku wierszy nie powinno powodować problemu).

Dla zainteresowanych

Popularne algorytmy imputacji danych często są oparte o algorytm najbliższych sąsiadów, czyli najbardziej podobne punkty. Innym podejściem, iteracyjnie imputującym wartości, jest algorytm MICE.

Skalowanie

Jest to bardzo ważny krok dla wielu modeli sztucznej inteligencji. Często takie modele mają pewne założenia co do danych wejściowych, a szczególnie popularnym założeniem jest, że wszystkie cechy mają wartości o podobnej skali. W szczególności regresja liniowa i logistyczna też czynią to założenie. Dlatego trzeba przeskalować nasze dane, żeby spełnić to założenie. Najprostsza metoda to `MinMaxScaler`, który przekształca wszystkie wartości do przedziału $[0, 1]$.

Istnieją też inne metody, np. standaryzacja, którą możesz pamiętać ze statystyki (jej wynikiem jest Z-score). Polega na odjęciu średniej i podzieleniu przez odchylenie standardowe każdej cechy. Wynikiem przekształcenia są cechy o średniej 0 i odchyleniu standardowym 1.

Więcej informacji na temat tego, dlaczego skalowanie jest tak istotne, możesz znaleźć tutaj.

Dla zainteresowanych

Porównanie różnych metod skalowania możesz znaleźć tutaj. Ciekawą metodą jest np. RobustScaler, który jest podobny do StandardScaler, ale używa mediany i kwartyli zamiast średniej i odchylenia standardowego. Są to tzw. robust statistics, czyli miary odporne na występowanie wartości odstających (outliers).

Przetwarzanie danych z wykorzystaniem Scikit-Learn

Mamy zatem do wykonania:

- na zmiennych numerycznych 2 operacje do wykonania: imputacja i skalowanie,
- na zmiennych kategorycznych: zastosowanie kodowania one-hot encoding.

W Scikit-learn służą do tego następujące klasy:

- `OneHotEncoder`, `SimpleImputer`, `MinMaxScaler` - transformacje, implementują metody `.fit()` i `.transform()`,
- `Pipeline` - do układania transformacji sekwencyjnie,
- `ColumnTransformer` - do układania transformacji równolegle, dla różnych kolumn.

Ważne: jako, że zaraz skorzystamy z regresji liniowej, do klasy `OneHotEncoder` trzeba przekazać `drop="first"`. Stworzy to 1 zmienną mniej, niż typowy one-hot encoding, np. `pd.get_dummies()`, gwarantując brak **idealnie współliniowych zmiennych (perfectly collinear features)**, co byłby niestabilny numerycznie. Dodatkowo, jako że przekształcamy już po podziale na zbiór treningowy i testowy, to możemy spotkać na zbiorze testowym nieliczne przypadki kategorii, których nie ma w zbiorze treningowym - kodujemy je wtedy po prostu jako wektory zer za pomocą `handle_unknown="ignore"`.

Na przykładzie `StandardScaler` (standaryzacja) rozpatrzmy, jak działają poszczególne metody.

Metoda `.fit()`

Do wykonania standaryzacji potrzebujemy dla każdej z cech określić 2 wartości - średnią oraz odchylenie standardowe. Formuła standaryzacji dla przypomnienia:

$$z = \frac{x - \mu}{\sigma}$$

Metodę `.fit()` wykonujemy tylko raz, dla **danych treningowych**. To powoduje, że obliczamy wartości μ oraz σ dla każdej cechy, na podstawie wartości ze zbioru treningowego. Wyuczone wartości zostają zapisane w obiekcie `StandardScaler` i mogą być później używane do przeprowadzenia standaryzacji zarówno dla danych treningowych, jak i testowych.

Co, gdyby dla danych testowych przeprowadzić osobną standaryzację?

Będziemy, na przykład, standaryzować kolumnę **GrLivArea** - powierzchnię nieruchomości. Załóżmy, że z danych treningowych wyszłoby, że średnia jest równa $60m^2$, a odchylenie standardowe - $20m^2$. Wtedy wartości z przedziału $[40, 80]$ zostaną przekształcone do $[-1, 1]$. Nasz model wykorzysta to przekształcenie i będzie uważał, że wartości po transformacji w pobliżu 0 oznaczają średniej wielkości apartamenty.

Określiliśmy parametry modelu i dostajemy kilkadziesiąt budynków z jakiejś zamożnej dzielnicy dla predykcji. Średnia powierzchnia dla tych budynków to około $160m^2$. Osobno przeprowadzając standaryzację dla takich danych testowych, zaburzylibyśmy rozkład tej cechy, gdyż tym razem wartości wokół 0 oznaczałyby dość duże mieszkania. Modele są niezwykle czułe na podobne zaburzenia - musimy przetwarzać dane spójnie, żeby nie doszło do podobnych sytuacji.

Czemu nie wywołać `.fit()` na wszystkich danych, a nie tylko treningowych?

Wydzieliliśmy dane testowe po to, żeby sprawdzać, jak model poradzi sobie z danymi, których do tej pory nigdy nie widział, bo to właśnie takie dane będzie on dostawać w praktyce, po wdrożeniu do realnego systemu. Ta ocena obejmuje też etap preprocessingu, w tym skalowania. Więc jeśli etap preprocessingu zobaczy dane testowe, to nie będziemy w stanie uczciwie estymować jego zachowania na nowych danych.

Wykorzystanie danych testowych w procesie treningu to błąd **wycieku danych (data leakage)**. Skutkuje on niepoprawnym, nadmiernie optymistycznym oszacowaniem jakości modelu.

Metoda `.transform()`

Przekształca dane za pomocą parametrów wyznaczonych w `.fit()`.

Metoda `.fit_transform()`

Metoda, która najpierw wykonuje `.fit()`, a potem `.transform()` i zwraca wynik ostatniej. W przypadku niektórych transformacji wykorzystuje ich specyfikę i działa szybciej, niż sekwencyjne wywołanie `.fit()` oraz `.transform()`. Trzeba jednak pamiętać, że możemy tego użyć tylko na zbiorze treningowym - na zbiorze testowym wywołujemy już tylko `.transform()`.

Zadanie 4 (0.5 punktu)

Stwórz pipeline'y dla zmiennych kategorycznych i numerycznych. Połącz je następnie z użyciem `ColumnTransformer`. "Wytrenuj" go na danych treningowych, a następnie przetransformuj dane treningowe oraz testowe.

Uwaga: przekaż do `ColumnTransformer` parametr `verbose_feature_names_out=False`, żeby nie zmieniał on nazw cech. Ułatwi nam to późniejszą analizę wyników.

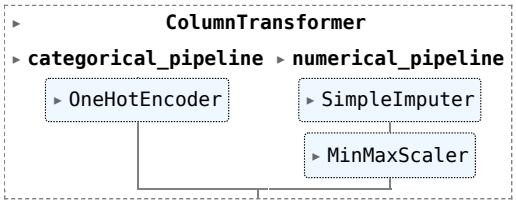
In [ ]:
```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler

one_hot_encoder = OneHotEncoder(
    drop="first", sparse_output=False, handle_unknown="ignore"
)
median_imputer = SimpleImputer(strategy="median")
min_max_scaler = MinMaxScaler()

categorical_pipeline = Pipeline(
    steps=[
        ('one_hot_encoder', one_hot_encoder)
    ]
)

numerical_pipeline = Pipeline(
    steps=[
        ('imputer', median_imputer),
        ('scaler', min_max_scaler)
    ]
)

column_transformer = ColumnTransformer(
    transformers=[
        ("categorical_pipeline", categorical_pipeline, categorical_features),
```

```
        ("numerical_pipeline", numerical_pipeline, numerical_features)
    ],
    verbose_feature_names_out=False)

X_train = column_transformer.fit_transform(X_train, y_train)
X_test = column_transformer.transform(X_test)
```

```
/home/dominiq/anaconda3/lib/python3.10/site-packages/sklearn/preprocessing/_encoders.py:202: UserWarning: Found unknown categories in columns [12, 15, 17] during transform. These unknown categories will be encoded as all zeros
  warnings.warn(
```

In [ ]:
```
column_transformer
```

Out[ ]:



Regresja liniowa

Możemy teraz przejść do przewidywania wartości domów. Naszym narzędziem będzie tutaj **regresja liniowa (linear regression)**, czyli model postaci:

$$\hat{y} = ax + b$$

gdzie $\hat{y}$ to zmienna zależna, x to zmienna niezależna (wartość cechy), a współczynniki obliczane są według wzorów opisanych tutaj, bez wątpienia znanych Ci z algebry liniowej i statystyki.

Rozwinięciem regresji liniowej jest wielokrotna regresja liniowa (*multiple linear regression*), która pozwala na wykorzystanie więcej niż jednej cechy do predykcji wartości. W takim modelu predykcja to kombinacja liniowa cech i wag, gdzie każda cecha posiada własną wagę. Więcej o tym mechanizmie możesz przeczytać tutaj. Formalnie jest to model postaci:

$$\hat{y} = \boldsymbol{w} \cdot \boldsymbol{x} + b = \sum_{i=1}^{d} w_i x_i + b$$

gdzie:

- d to **wymiarowość (dimensionality)**, czyli liczba cech
- $\boldsymbol{w}$ to wektor wag o długości d
- w_i to wagi poszczególnych cech
- b to **wyraz wolny (bias / intercept)**, punkt przecięcia ze środkiem układu współrzędnych

Pozostaje pytanie, jak wyznaczyć wagi $\boldsymbol{w}$ i wyraz wolny b. Można to robić na różne sposoby, przy czym klasyczna regresja liniowa minimalizuje **błąd średniokwadratowy (mean squared error, MSE)**. Jest to przykład **funkcji kosztu (loss function / cost function)**, a konkretnie *squared loss / L2 loss*. Ma on postać:

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2$$

gdzie $\hat{y}$ to wartość przewidywana przez model, y - prawdziwa, a n to liczba punktów w zbiorze.

W Scikit-learn ten model implementuje klasa `LinearRegression`. Jej ważne cechy:

- domyślnie uwzględnia intercept (bias) przez `fit_intercept=True`; jeżeli nasze dane są już wycentrowane, to jest to niepotrzebne i może powodować problemy numeryczne,
- używa implementacji z pseudoodwrotnością Moore'a-Penrose'a (SVD),
- nie pozwala na regularyzację, do tego trzeba użyć innych klas.

Jak ocenić, jak taki model sobie radzi? Trzeba tutaj użyć pewnej **metryki (metric)**, czyli wyznacznika jakości modelu. Można na to patrzeć z wielu różnych perspektyw, w zależności od charakterystyki problemu. Tradycyjnie używa się **Root MSE (RMSE)**, czyli pierwiastka kwadratowego z MSE. Ma ważne zalety:

- regresja liniowa z definicji modelu optymalizuje miarę MSE, więc używamy metryki dobrze związanej z modelem,
- dzięki pierwiastkowaniu ma tę samą jednostkę, co przewidywane wartości. .

Jest też dość czuła na wartości odstające, ale może to być korzystne, w zależności od zastosowania.

$$RMSE(y, \hat{y}) = \sqrt{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

W Scikit-learn RMSE liczy się dość specyficznie, bo używa się funkcji do MSE z argumentem `squared=False`.

Dla zainteresowanych

Minimalizując inne rodzaje błędu, otrzymujemy modele liniowe o innych parametrach, ale tej samej postaci funkcji. Typowo modele te są bardziej odporne na wartości odstające, ale bardziej kosztowne w treningu. Są to np. quantile regression optymalizująca koszt L1 (*mean absolute error*) czy Huber regression, optymalizująca tzw. Huber loss (połączenie L1 i L2).

Obliczanie regresji liniowej używa pseudoodwrotności Moore'a-Penrose'a i SVD. Objaśnia to dobrze ten tutorial.

In [ ]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LinearRegression

# all variables are in range [0, 1], so we don't need an intercept
reg_linear = LinearRegression(fit_intercept=False)
reg_linear.fit(X_train, y_train)

y_pred = reg_linear.predict(X_test)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print(f"RMSE: {rmse:.4f}")
```

```
RMSE: 0.1160
```

Czy taki błąd to duży, czy mały? Wszystko zależy od skali wartości przewidywanych. Trzeba pamiętać, że dokonaliśmy logarytmowania zmiennej docelowej, więc trzeba to sprawdzić po transformacji odwrotnej `np.expm1`. Po tej operacji wartość błędu będzie wyrażona w dolarach.

Zbyt małe i nadmierne dopasowanie

W trakcie trenowania modelu może dojść do sytuacji, w której zostanie on **przeuczony (overfitting)**. W takim wypadku model nadmiernie dostosowuje się do danych treningowych, "zakuwając" je. Daje wtedy bardzo dokładne wyniki na zbiorze treningowym, ale kiepskie na zbiorze testowym. Modele przeuczone słabo zatem się **generalizują (generalization)**.

Dlatego wcześniej wydzieliliśmy zbiór testowy, za pomocą którego oceniamy skuteczność naszego modelu. Pozwala to uniknąć powyższego błędu. Przeuczenie bardzo często można rozpoznać właśnie po różnym zachowaniu modelu na danych treningowych i testowych. Jeśli z danymi treningowymi model radzi sobie dużo lepiej, niż z testowymi, to istnieje dużo ryzyko, że model został przeuczony i skupił się na zapamiętywaniu konkretnych przykładów, na których się uczył, niż na wyciąganiu z nich uniwersalnych wzorców. Taki model słabo się generalizuje i nie poradzi sobie z nowymi danymi.

Sprawdza się to następująco:

- obliczamy błąd treningowy oraz testowy,
- jeżeli oba błędy są wysokie, to mamy zbyt małe dopasowanie (*underfitting*) i trzeba użyć pojemniejszego modelu,
- jeżeli błąd treningowy jest dużo niższy od testowego, to mamy nadmierne dopasowanie (*overfitting*) i model trzeba regularyzować.

W praktyce paradoksalnie często model o większej pojemności z mocną regularyzacją działa lepiej od prostszego modelu ze słabą regularyzacją. Wyjaśnianie, czemu tak jest, to otwarty problem naukowy, szczególnie w kontekście sieci neuronowych.

Przeuczenie modelu jest bardzo istotnym problemem w sztucznej inteligencji i istnieje szereg metod, służących zapobieganiu tego zjawiska. Jedną z nich jest regularyzacja - do globalnej funkcji błędu dodawane są "kary" za tworzenie zbyt złożonych modeli. Typowe metody regularyzacji to L1 oraz L2, które penalizują wielkość parametrów obliczonych w trakcie treningu. Obie te wartości są tak naprawdę normami (odpowiednio `l1` i `l2`) wektorów wag modelu, przeskalowanymi przez określoną wartość. Dodawanie tych kar ma zapobiec przeuczeniu, bo typowo duże wagi w regresji liniowej i podobnych modelach oznaczają przeuczenie.

Czemu tak jest? Przeuczenie bierze się z tego, że nasz model "zakuwa" zbiór treningowy, ucząc się **szumu (noise)** w danych, przypisując nadmierne znaczenie niewielkim różnicom w wartościach cech. Jeżeli cecha ma dużą wagę, to nawet niewielka zmiana jej wartości bardzo zmienia finalną predykcję (która jest kombinacją liniową). Dzięki regularyzacji, jeżeli model podczas treningu będzie chciał zwiększyć wagę dla cechy, to musi mu się to opłacać. Innymi słowy, zwiększenie wagi cechy musi zmniejszyć koszt (np. MSE) bardziej, niż wzrośnie kara z regularyzacji.

Jak słusznie się domyślić, zbyt duże kary spowoduję z kolei niedouczenie (ang. *underfitting*). Więcej o konstrukcji i zastosowaniach regularyzacji L1 i L2 możesz przeczytać tutaj.

Dla zainteresowanych

W praktyce detekcja nadmiernego dopasowania nie musi być wcale taka oczywista. Nasz model może przeuczać się tylko na niektórych segmentach danych, dla nietrywialnych kombinacji cech etc. Testowanie modeli ML i detekcja overfittingu jest otwartym problemem badawczym, ale powstają już pierwsze narzędzia do tego, np. Giskard.

Zadanie 5 (1.0 punkt)

Uzupełnij kod funkcji `assess_regression_model` o:

- obliczenie predykcji na zbiorze treningowym oraz testowym,
- transformacje eksponencjalne, żeby wrócić do oryginalnej jednostki (dolara),
- obliczenie RMSE dla zbioru treningowego i testowego,
- wypisywanie RMSE, zaokrąglonego do 2 miejsc po przecinku.

Skomentuj wyniki. Czy następuje przeuczenie modelu? Oceń także sam błąd, czy subiektywnie to duża wartość, biorąc pod uwagę rozkład zmiennej docelowej (wartości i wykresy w sekcji EDA)?

In [ ]:
```python
def assess_regression_model(model, X_train, X_test, y_train, y_test) -> None:
    # predict for train and test
    y_pred_train    = model.predict(X_train)
    y_pred_test     = model.predict(X_test)

    # exponential transform for y_train, y_test and predictions
    y_pred_test     = np.expm1(y_pred_test)
    y_pred_train    = np.expm1(y_pred_train)
    y_test          = np.expm1(y_test)
    y_train         = np.expm1(y_train)

    # handle passible overflow in y_pred_test
    y_pred_test = [elem if not np.isinf(elem) else np.ma.masked_invalid(y_pred_test).mean() for elem in

    # calculate train and test RMSE
    rmse_train  = mean_squared_error(y_train, y_pred_train, squared=False)
    rmse_test   = mean_squared_error(y_test, y_pred_test, squared=False)

    # print train and test RMSE
    print(f"[TRAIN] RMSE: {rmse_train:.2f}")
    print(f"[TEST] RMSE: {rmse_test:.2f}")
```

In [ ]:
```python
assess_regression_model(reg_linear, X_train, X_test, y_train, y_test)
```

```
[TRAIN] RMSE: 16745.60
[TEST] RMSE: 21323.38
```

Wartość `RMSE` dla danych testowych jest prawie o 30% więsza od wartości dla danych treningowych. Istnieje szansa że nastąpiło przeuczenie modelu. Same wartości nie są jednak aż tak duże.

Regresja regularyzowana (ridge, LASSO)

Regularyzacja zmniejsza pojemność modelu regresji liniowej, narzucając mniejsze wagi poprzez penalizację dużych wag w funkcji kosztu. Regresja liniowa z regularyzacją L2 nazywa się *ridge regression*, z regularyzacją L1 - *LASSO regression*, a z oboma naraz - *ElasticNet regression*. Formalnie mamy:

$$L_{ridge}(y, \hat{y}) = \frac{1}{n}(y - \hat{y})^2 + \lambda||\boldsymbol{w}||_2^2$$

$$L_{LASSO}(y, \hat{y}) = \frac{1}{n}(y - \hat{y})^2 + \alpha||\boldsymbol{w}||_1$$

$$L_{ElasticNet}(y, \hat{y}) = \frac{1}{n}(y - \hat{y})^2 + \lambda||\boldsymbol{w}||_2^2 + \alpha||\boldsymbol{w}||_1$$

Jak widać, regularyzacja dodaje do zwykłego kosztu MSE dodatkowe wyrazy, penalizujące wielkość wag $\boldsymbol{w}$. **Siłę regularyzacji (regularization strength)**, czyli jak mocna jest taka kara, wyznacza współczynnik, oznaczany typowo λ albo α. Jest to **hiperparametr (hyperparameter)**, czyli stała modelu, którą narzucamy z góry, przed treningiem. Nie jest on uczony z danych. Jak go dobrać, omówimy poniżej.

Regresja ridge (L2) zmniejsza wagi i jest różniczkowalna (szybsza i łatwiejsza w treningu). Regresja LASSO (L1) dokonuje **selekcji cech (feature selection)**, zmniejszając często wagi cech dokładnie do zera, eliminując tym samym słabe cechy. Oba naraz realizuje model ElasticNet.

W Scikit-learn implementują je klasy `Ridge`, `Lasso` oraz `ElasticNet`. Najważniejszy hiperparametr każdego z tych modeli to siła regularyzacji, która we wszystkich klasach to `alpha`. Scikit-learn definiuje regularyzację ElasticNet dość specyficznie, za pomocą parametru `l1_ratio`, który wyznacza, jaki ułamek siły regularyzacji przypada dla L1, a jaki dla L2:

$$L_{ElasticNet}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2 + \alpha \cdot (1 - L1\_ratio) \cdot ||\boldsymbol{w}||_2^2 + \alpha \cdot L1\_ratio \cdot ||\boldsymbol{w}||_1$$

Inne ważne uwagi:

- liczba iteracji `max_iter` wyznacza liczbę iteracji solwera; im więcej, tym dokładniejsze rozwiązanie, ale tym dłuższy czas obliczeń,
- jeżeli `max_iter` będzie zbyt mała i algorytm nie osiągnie zbieżności, to dostaniemy ostrzeżenie, wtedy zwykle trzeba po prostu ją zwiększyć, np. 10-krotnie,
- jeżeli nie potrzebujemy bardzo precyzyjnego rozwiązania, można ustawić większe `tol` dla przyspieszenia obliczeń.

Jako że nasz model jest regularyzowany i nie ma ryzyka problemów numerycznych, to teraz już obliczamy intercept.

In [ ]:
```
from sklearn.linear_model import Ridge, Lasso

reg_ridge = Ridge(random_state=0)
reg_lasso = Lasso(random_state=0)

reg_ridge.fit(X_train, y_train)
reg_lasso.fit(X_train, y_train)

assess_regression_model(reg_ridge, X_train, X_test, y_train, y_test)
print()
assess_regression_model(reg_lasso, X_train, X_test, y_train, y_test)
```

```
[TRAIN] RMSE: 16867.30
[TEST] RMSE: 18880.97

[TRAIN] RMSE: 79579.79
[TEST] RMSE: 80091.99
```

W przypadku regularyzacji L2 domyślna siła regularyzacji (`alpha=1.0`) znacząco poprawiła wynik, natomiast w przypadku L1 mamy bardzo silny underfitting.

Tuning hiperparametrów, zbiór walidacyjny

Praktycznie wszystkie modele ML mają hiperparametry, często liczne, które w zauważalny sposób wpływają na wyniki, a szczególnie na underfitting i overfitting. Ich wartości trzeba dobrać zatem dość dokładnie. Jak to zrobić? Proces doboru hiperparametrów nazywa się **tuningiem hiperparametrów** (*hyperparameter tuning*).

Istnieje na to wiele sposobów. Większość z nich polega na tym, że trenuje się za każdym razem model z nowym zestawem hiperparametrów i wybiera się ten zestaw, który pozwala uzyskać najlepsze wyniki. Metody głównie różnią się między sobą sposobem doboru kandydujących zestawów hiperparametrów.

Najprostsze i najpopularniejsze to:

- **pełne przeszukiwanie** (*grid search*) - definiujemy możliwe wartości dla różnych hiperparametrów, a metoda sprawdza ich wszystkie możliwe kombinacje (czyli siatkę),
- **losowe przeszukiwanie** (*randomized search*) - definiujemy możliwe wartości jak w pełnym przeszukiwaniu, ale sprawdzamy tylko ograniczoną liczbę losowo wybranych kombinacji.

Jak ocenić, jak dobry jest jakiś zestaw hiperparametrów? Nie możemy sprawdzić tego na zbiorze treningowym - wyniki byłyby zbyt optymistyczne. Nie możemy wykorzystać zbioru testowego - mielibyśmy data leakage, bo wybieralibyśmy model explicite pod nasz zbiór testowy. Trzeba zatem osobnego zbioru, na którym będziemy na bieżąco sprawdzać jakość modeli dla różnych hiperparametrów. Jest to **zbiór walidacyjny** (*validation set*).

Zbiór taki wycina się ze zbioru treningowego. Dzielimy zatem nasze dane nie na dwie, ale trzy części: treningową, walidacyjną i testową. Typowe proporcje to 60-20-20% lub 80-10-10%.

Metody tuningu hiperparametrów są zaimplementowane w Scikit-Learn jako `GridSearchCV` oraz `RandomizedSearchCV`. Są też bardziej wyspecjalizowane metody dla konkretnych modeli, które są dla nich typowo o wiele szybsze.

Uwaga: warto zauważyć, że liczba możliwych kombinacji rośnie gwałtownie wraz z liczbą hiperparametrów i ich możliwych wartości. Mając siatkę na 3 hiperparametry po 10 możliwych wartości dla każdego, otrzymujemy 1000 możliwych kombinacji. W pracy w ML płacą nam też za to, że wiemy, jakie siatki dobrać :)

Dla zainteresowanych

Szczególnie inteligentne są metody tuningu z grupy metod optymalizacji bayesowskiej (Bayesian hyperparameter optimization / Bayesian HPO). Są to np. procesy Gaussowskie oraz Tree Parzen Estimator (TPE). Wykorzystują one dość zaawansowaną statystykę, aby zamodelować, jak poszczególne hiperparametry wpływają na wynik i dobierają takie kolejne kombinacje hiperparametrów, które są ich zdaniem najbardziej obiecujące. W szczególności wiele z tych metod traktuje dobór hiperparametrów jak problem regresji, gdzie parametrami są hiperparametry modelu, które dobieramy.

Takich metod szczególnie często używa się przy tuningu hiperparametrów dla sieci neuronowej, gdyż jej wytrenowanie jest czasochłonne, a więc nie możemy pozwolić sobie na sprawdzenie licznych kombinacji, bo zbyt dużo by nas to kosztowało.

Ta metoda została zaimplementowana w wielu frameworkach, jak np. Optuna czy Hyperopt. Więcej można o nich przeczytać tutaj.

Walidacja skrośna

Jednorazowy podział zbioru na części nazywa się *split validation* lub *holdout*. Używamy go, gdy mamy sporo danych, i 10-20% zbioru jako dane walidacyjne czy testowe to dość dużo, żeby mieć przyzwoite oszacowanie. Zbyt mały zbiór walidacyjny czy testowy da nam mało wiarygodne wyniki - nie da się nawet powiedzieć, czy zbyt pesymityczne, czy optymistyczne! W praktyce niestety często mamy mało danych. Trzeba zatem jakiejś magicznej metody, która stworzy nam więcej zbiorów walidacyjnych z tej samej ilości danych.

Taką metodą jest **walidacja skrośna** (*cross-validation, CV*). Polega na tym, że dzielimy zbiór na K równych podzbiorów, tzw. *foldów*. Każdy podzbiór po kolei staje się zbiorem walidacyjnym, a pozostałe łączymy w zbiór treningowy. Przykładowo, jeżeli mamy 5 foldów (1, 2, 3, 4, 5), to będziemy mieli po kolei:

- zbiór treningowy: (2, 3, 4, 5), walidacyjny: (1)
- zbiór treningowy: (1, 3, 4, 5), walidacyjny: (2)
- zbiór treningowy: (1, 2, 4, 5), walidacyjny: (3)
- zbiór treningowy: (1, 2, 3, 5), walidacyjny: (4)
- zbiór treningowy: (1, 2, 3, 4), walidacyjny: (5)

Trenujemy zatem K modeli dla tego samego zestawu hiperparametrów i każdy testujemy na zbiorze walidacyjnym. Mamy K wyników dla zbiorów walidacyjnych, które możemy uśrednić (i ew. obliczyć odchylenie standardowe). Takie wyniki są znacznie bardziej wiarygodne zgodnie ze statystyką (moc statystyczna itp.). Typowo używa się 5 lub 10 foldów, co jest dobrym balansem między liczbą modeli do wytrenowania i wielkością zbiorów walidacyjnych.

Szczególnym przypadkiem jest Leave-One-Out Cross-Validation (LOOCV), w którym ilość podzbiorów (*foldów*) jest równa ilości rekordów. Czyli w danej chwili tylko 1 przykład jest zbiorem walidacyjnym. Daje to możliwość prawie całkowitego wykorzystania naszych danych (w każdej iteracji musimy wydzielić tylko 1 przykład na zbiór walidacyjny, cała reszta jest naszym zbiorem treningowym), ale wprowadza ogromny koszt obliczeniowy. Jest to opłacalne tylko w szczególnych przypadkach.

Można zauważyć, że w nazwach klas do tuningu parametrów, wspomnianych wyżej, mamy sufiks `CV` - to jest właśnie *Cross Validation*.

Dla zainteresowanych

Walidacji skrośnej można użyć także do testowania, tworząc wiele zbiorów testowych. Można połączyć obie techniki, co daje tzw. nested cross-validation. Jest to bardzo kosztowna, ale jednocześnie bardzo precyzyjna technika.

RidgeCV, LassoCV, ElasticNetCV

W przypadku regresji liniowej istnieją bardzo wydajne implementacje walidacji skrośnej, głównie dzięki prostocie tego modelu. W Scikit-learn są to odpowiednio `RidgeCV`, `LassoCV` oraz `ElasticNetCV`.

`RidgeCV` domyślnie wykorzystuje efektywną implementację Leave-One-Out Cross-Validation (LOOCV). Jest to możliwe dzięki pewnym sztuczkom opartym na algebrze liniowej, wyjaśnionych w dokumentacji w kodzie (dla zainteresowanych). Co ważne, jest to operacja o wiele szybsza niż osobne grid search + ridge regression, a nawet od `RidgeCV` z mniejszą liczbą foldów.

`LassoCV` oraz `ElasticNetCV` iterują od najmniejszych do największych wartości `alpha` (siły regularyzacji), używając rozwiązania dla mniejszej siły regularyzacji jako punktu początkowego dla kolejnej wartości. Odpowiada to po prostu dość inteligentnemu wyborowi punktu startowego w optymalizacji funkcji kosztu, a znacznie obniża koszt obliczeniowy.

Zadanie 6 (1.0 punkt)

Użyj klas `RidgeCV` oraz `LassoCV` do tuningu hiperparametrów.

Dla `RidgeCV` sprawdź 1000 wartości `[0.1, 100]` w skali liniowej - przyda się `np.linspace()`. Użyj LOOCV.

Dla `LassoCV` Scikit-learn sam dobierze wartości, musisz podać tylko liczbę wartości alfa do sprawdzenia - użyj 1000. Użyj 5-fold CV. Pamiętaj o podaniu `random_state=0` - solver jest niedeterministyczny.

Wypisz znalezione optymalne wartości siły regularyzacji `.alpha_` dla obu modeli, zaokrąglone do 4 miejsca po przecinku dla czytelności.

__Ciekawostka__

Atrybuty z `_` (*underscore*) na końcu w Scikit-Learn oznaczają, że zostały one wyliczone podczas treningu (`.fit()`). W powyższym przypadku optymalny współczynnik regularyzacji `.alpha_` został wyznaczony dopiero po przeprowadzeniu tuningu hiperparametrów.

Jeśli zajrzeć do dokumentacji dla klasy `LinearRegression`, to można zauważyć takie atrybuty jak `.coef_` przechowujący wyznaczone współczynniki cech, czy `.intercept_` - wyraz wolny.

Takie atrybuty pozwalają przeprowadzić dogłębniejszą analizę wytrenowanego modelu.

Przetestuj modele z użyciem `assess_regression_model()`. Skomentuj wyniki. Czy udało się wyeliminować overfitting?

In [ ]:
```
from sklearn.linear_model import RidgeCV, LassoCV

reg_ridge_ = RidgeCV(alphas=np.linspace(0.1, 100, num=1000)).fit(X_train, y_train)
reg_lasso_ = LassoCV(n_alphas=1000, cv=5, random_state=0).fit(X_train,y_train)

print(f"[RIDGE]: {reg_ridge_.alpha_:.4f}")
print(f"[LASSO]: {reg_lasso_.alpha_:.4f}")
```

```
[RIDGE]: 2.9000
[LASSO]: 0.0003
```

In [ ]:
```
from sklearn.linear_model import Ridge, Lasso

reg_ridge = Ridge(random_state=0, alpha=reg_ridge_.alpha_)
reg_lasso = Lasso(random_state=0, alpha=reg_lasso_.alpha_)

reg_ridge.fit(X_train, y_train)
reg_lasso.fit(X_train, y_train)

assess_regression_model(reg_ridge, X_train, X_test, y_train, y_test)
print()
assess_regression_model(reg_lasso, X_train, X_test, y_train, y_test)
```

```
[TRAIN] RMSE: 17213.15
[TEST] RMSE: 18769.18

[TRAIN] RMSE: 18112.62
[TEST] RMSE: 18693.99
```

Wartości RMSE po tuningu hiperparametrów dla zbiorów testowych i treningowych są bardzo zbliżone. Możemy więc założyć że udało się wyeliminować overfitting.

Regresja wielomianowa

Regresja wielomianowa to po prostu dodanie wielomianów cech do naszych danych:

$$[a, b, c, d] -> [a, b, c, d, a^2, b^2, c^2, d^2, ab, ac, ad, bc, bd, cd]$$

Pozwala to na uwzględnienie bardziej złożonych kombinacji cech, których sama regresja liniowa, ze względu na swoją prostotę, nie jest w stanie uwzględnić.

W Scikit-learn regresja wielomianowa składa się z 2 osobnych kroków: wygenerowania cech wielomianowych i użycia zwykłej regresji liniowej. Pozwala to na użycie tej transformacji dla dowolnych algorytmów, nie tylko regresji liniowej.

Kwestią sporną jest, czy jest sens przeprowadzać taką transformację dla zmiennych po one-hot encodingu. Potęgi na pewno nie mają sensu, natomiast interakcje realizują po prostu operację koniunkcji (AND), ale łatwo prowadzi to do eksplozji wymiarowości. Dla uproszczenia poniżej zastosujemy transformację dla wszystkich cech.

Warto pamiętać, że jeżeli używamy modelu, który sam dodaje intercept (jak regresja liniowa), to trzeba przekazać `include_bias=False`. Żeby wymiarowość zbytnio nam nie urosła, użyjemy `interaction_only=True`.

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
poly_features.fit(X_train)

X_train_poly = poly_features.transform(X_train)
X_test_poly = poly_features.transform(X_test)

reg_ridge_cv_poly = RidgeCV(alphas=np.linspace(0.1, 100, 1000))
reg_ridge_cv_poly.fit(X_train_poly, y_train)

assess_regression_model(reg_ridge_cv_poly, X_train_poly, X_test_poly, y_train, y_test)
print()
print(f"Ridge + polynomial features alpha: {reg_ridge_cv_poly.alpha_:.4f}")
```

```
[TRAIN] RMSE: 12963.36
[TEST] RMSE: 18285.73

Ridge + polynomial features alpha: 86.1000
```

Co ciekawe, model bardziej zbliżył się do przeuczenia, ale błąd testowy zmalał. Jest to niezbyt częste, ale możliwe.

Regresja logistyczna

Regresja logistyczna jest modelem, który pozwala na przewidywanie wartości zmiennych dychotomicznych w oparciu o jedną lub większą ilość cech. Funkcją bazową regresji logistycznej jest funkcja logistyczna. Bardzo ciekawe podsumowanie dotyczące matematyki stojącej za regresją logistyczną znajdziesz tu.

Do klasyfikacji wykorzystamy zbiór Bank Marketing, w którym przewiduje się, czy dana osoba będzie zainteresowana lokatą terminową w banku. Precyzyjny targetowany marketing jest ważny z perspektywy biznesu, bo w praktyce chce się reklamować tak mało, jak to możliwe. Bank zarabia tylko na tych osobach, które są faktycznie zainteresowane reklamą, a pozostałych można łatwo zrazić zbyt dużą liczbą reklam, więc precyzyjna ocena przynosi tu realne zyski.

Zbiór posiada dwie wersje, uproszczoną oraz rozszerzoną o dodatkowe atrybuty socjoekonomiczne (np. sytuację ekonomiczną w planowanym momencie reklamy). Wykorzystamy tę drugą, bo są to bardzo wartościowe cechy. Dodatkowo każda wersja posiada pełen zbiór (ok. 45 tysięcy przykładów) oraz pomniejszony (ok. 4 tysiąca przykładów). Dzięki skalowalności regresji logistycznej możemy bez problemu wykorzystać pełny zbiór z dodatkowymi cechami.

Opisy zmiennych znajdują się w pliku `bank_marketing_description.txt`.

Zadanie 7 (1.0 punkt)

Wczytywanie i czyszczenie danych

1. Załaduj zbiór danych z pliku `bank_marketing_data.csv` do DataFrame'a. Zwróć uwagę, że separatorem jest średnik (argument `sep`).
2. Usuń kolumny:
   - `default`, czy klient ma zadłużenie na karcie kredytowej; ma tylko 3 wartości `yes`
   - `duration`, czas trwania ostatniego telefonu reklamowego; autorzy sugerują usunięcie w opisie zbioru, bo nie znamy tej wartości przed wykonaniem telefonu
   - `pdays`, liczba dni od ostatniego telefonu reklamowego w ramach danej kampanii marketingowej; jeżeli to pierwszy kontakt, to wartość to 999, i ciężko byłoby włączyć taką cechę do modelu, a mamy już i tak informację o tym, czy to pierwszy kontakt z klientem w zmiennej `previous`
   - `poutcome`, wynik poprzedniej kampanii; w zdecydowanej większości przypadków to `nonexistent`
3. Dokonaj filtrowania wierszy:
   - usuń wiersze z `education` na poziomie `illiterate`, jest ich tylko kilkanaście
4. Zakoduj odpowiednio zmienne `education`, `contact`, `month`, `day_of_week` i `y`. Dla ułatwienia słowniki są w zmiennych poniżej.
5. Wyodrębnij kolumnę `y` do zmiennej `y` (pamiętaj o usunięciu jej z DataFrame'a).

In [ ]:
```
education_mapping = {
    "basic.4y": "primary",
    "basic.6y": "primary",
    "basic.9y": "primary",
    "high.school": "secondary",
    "professional.course": "secondary",
    "university.degree": "tertiary",
}

contact_mapping = {
    "telephone": 0,
    "cellular": 1,
}
```

```
month_mapping = {
    "jan": 1,
    "feb": 2,
    "mar": 3,
    "apr": 4,
    "may": 5,
    "jun": 6,
    "jul": 7,
    "aug": 8,
    "sep": 9,
    "oct": 10,
    "nov": 11,
    "dec": 12,
}

day_of_week_mapping = {
    "mon": 1,
    "tue": 2,
    "wed": 3,
    "thu": 4,
    "fri": 5,
}

y_mapping = {
    "no": 0,
    "yes": 1,
}
```

In [ ]:
```
df = pd.read_csv("bank_marketing_data.csv", sep=";", na_values="unknown")

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             40858 non-null  object 
 2   marital         41108 non-null  object 
 3   education       39457 non-null  object 
 4   default         32591 non-null  object 
 5   housing         40198 non-null  object 
 6   loan            40198 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [ ]:
```
df = df.drop(["default", "duration", "pdays", "poutcome"], axis="columns")
```

In [ ]:
```
df = df.loc[~df["education"].isin(["illiterate"]), :]
```

In [ ]:
```
df = df.replace(
    {
        "education" : education_mapping,
        "contact": contact_mapping,
        "month": month_mapping,
        "day_of_week": day_of_week_mapping,
        "y" : y_mapping
    }
)
```

In [ ]:
```
y = df.pop("y")
```

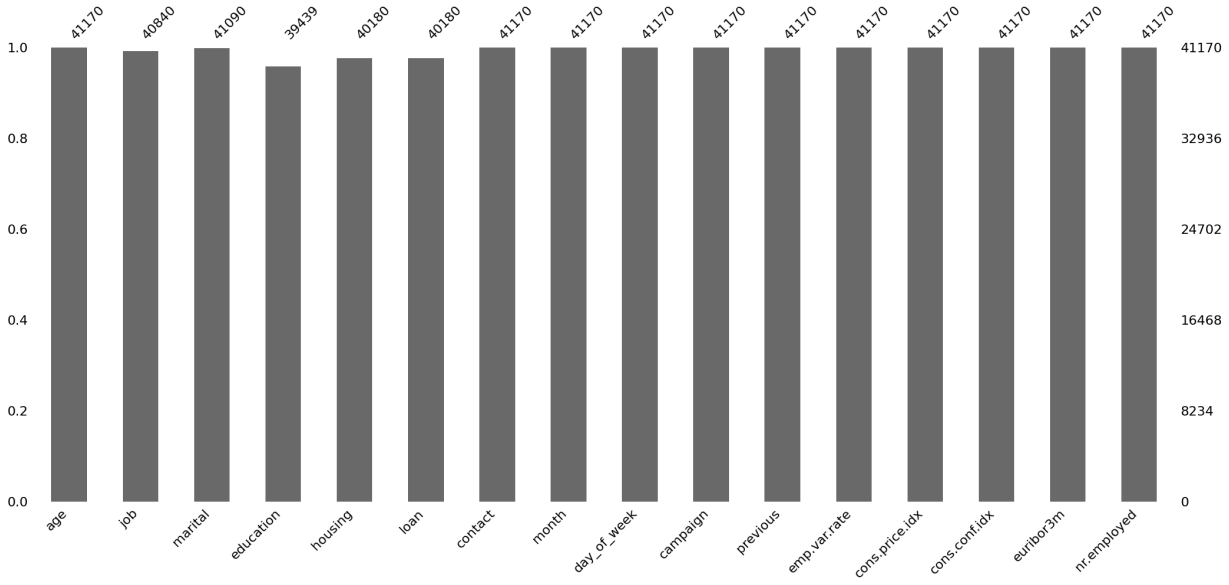
Zadanie 8 (0.5 punktu)

Exploratory Data Analysis (EDA)

1. Sprawdź, czy są jakieś wartości brakujące za pomocą biblioteki `missingno`. Jeżeli tak, to sprawdź w dokumentacji zbioru, jaka byłaby sensowna wartość do ich uzupełnienia.
2. Narysuj wykres (bar plot) z częstością klas. Uwzględnij częstość na wykresie (to może się przydać). Pamiętaj o tytule i opisaniu osi.

In [ ]:
```
from random import random

# plot missing values
msno.bar(df)
```

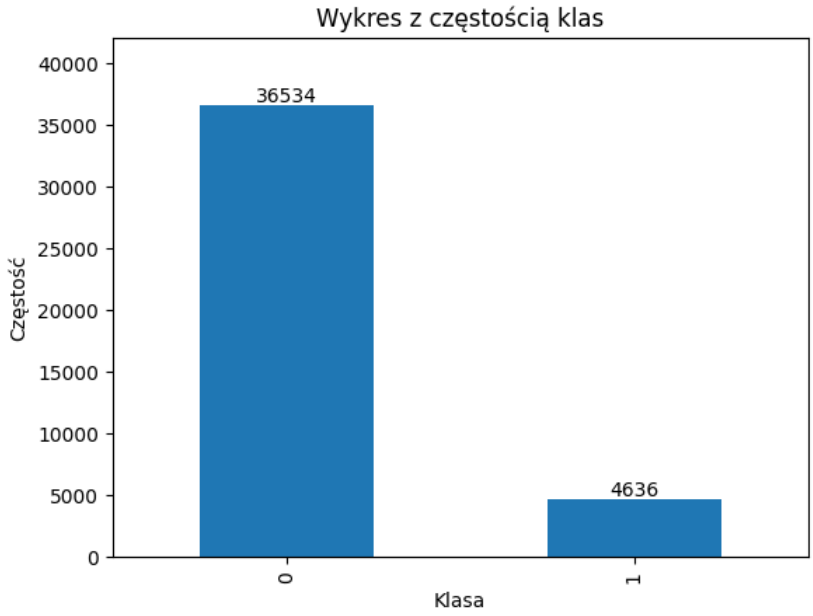
Out[ ]:
```
<Axes: >
```



In [ ]:
```
# job: Unknown most likely means unemployed
replace_na(df, "job", value="unemployed")

# marital: Unknown most likely means single
replace_na(df, "marital", value="single")

# education: School compulsion requires getting secondary education
replace_na(df, "education", value="secondary")

# Unknown loan status most likely means no loan
replace_na(df, "housing", value="no")
replace_na(df, "loan", value="no")

# contact: Unknown most likely means no contact
replace_na(df, "contact", value=0)
replace_na(df, "campaign", value=0)
replace_na(df, "previous", value=0)

# nr.employed: Unknown number of employees most likely means no employees
replace_na(df, "nr.employed",   value=0)
```

In [ ]:
```
y_plot = y.value_counts().plot.bar()
y_plot.margins(0.15, 0.15)
y_plot.bar_label(y_plot.containers[0])
y_plot.set_title("Wykres z częstością klas")
y_plot.set_xlabel("Klasa")
y_plot.set_ylabel("Częstość")
```

Out[ ]:
```
Text(0, 0.5, 'Częstość')
```



Jak widać, będziemy tu mieli do czynienia z problemem klasyfikacji niezbalansowanej. Na szczęście funkcja kosztu w regresji logistycznej pozwala na dodanie **wag klas (class weights)**, aby przypisać większą wagę interesującej nas klasie pozytywnej. Scikit-learn dla wartości `class_weights="balanced"` obliczy wagi odwrotnie proporcjonalne do częstości danej klasy w zbiorze.

Zadanie 9 (1.0 punkt)

Podział i preprocessing danych

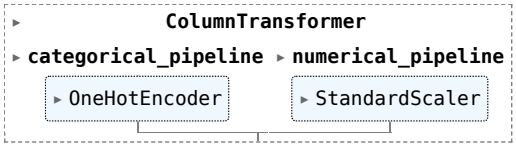
1. Dokonaj podziału zbioru na treningowy i testowy w proporcjach 75%-25%. Pamiętaj o użyciu podziału ze stratyfikacją (argument `stratify`), aby zachować proporcje klas. Ustaw `random_state=0`.
2. Stwórz `ColumnTransformer`, przetwarzający zmienne kategoryczne za pomocą `OneHotEncoder` (teraz już nie musimy robić `drop="first"`), a numeryczne za pomocą `StandardScaler`. Zaaplikuj go do odpowiednich kolumn.

In [ ]:
```
categorical_features = df.select_dtypes(include="object").columns
numerical_features = df.select_dtypes(exclude="object").columns

X_train, X_test, y_train, y_test = train_test_split(
    df, y, test_size=0.25, train_size=0.75, random_state=0, stratify=y
)
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler

standard_scaler = StandardScaler()

one_hot_encoder = OneHotEncoder(
    sparse_output=False, handle_unknown="ignore"
)
median_imputer = SimpleImputer(strategy="median")

categorical_pipeline = Pipeline(
    steps=[
        ('one_hot_encoder', one_hot_encoder)
    ]
)

numerical_pipeline = Pipeline(
    steps=[
        ('scaler', standard_scaler),
    ]
)

column_transformer = ColumnTransformer(
    transformers=[
        ("categorical_pipeline", categorical_pipeline, categorical_features),
        ("numerical_pipeline", numerical_pipeline, numerical_features)
    ],
    verbose_feature_names_out=False)

X_train = column_transformer.fit_transform(X_train, y_train)
X_test = column_transformer.transform(X_test)
```

In [ ]:
```
column_transformer
```

Out[ ]:



Metryki klasyfikacji binarnej

W klasyfikacji binarnej mamy tylko dwie klasy, z konwencji oznaczamy jedną klasę jako negatywną, a drugą - pozytywną. W naszym przypadku klasą negatywną będą osoby niezainteresowane lokatą - nie chcemy im pokazywać naszych reklam, bo to będzie raczej bezskutecznie, a reklama kosztuje. Naszym targetem będą osoby oznaczone klasą pozytywną.

Wytrenowaliśmy model, ale jak sprawdzić jakość jego działania? Metryki z regresji raczej za wiele nam nie pomogą. Potrzebujemy zdefiniować nowe.

Celność (*Accuracy*)

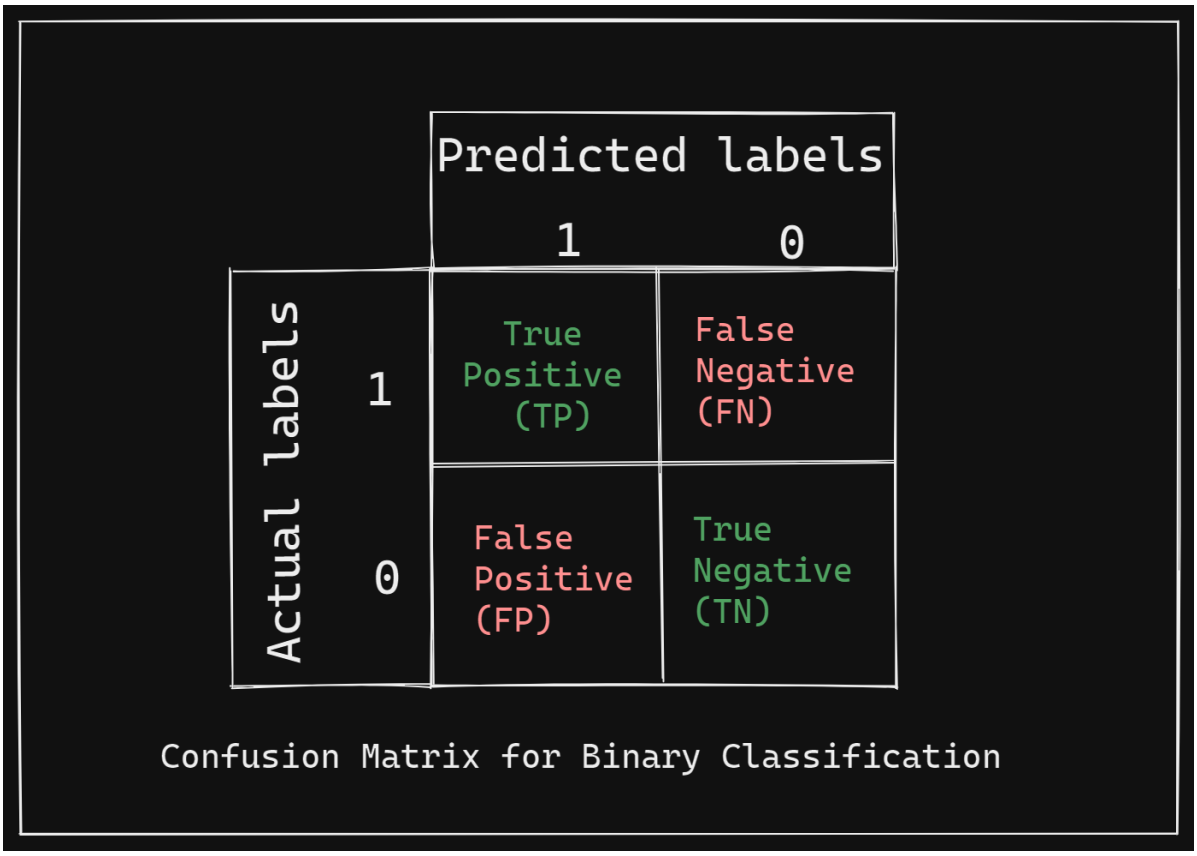
Najprostszym sposobem oceny klasyfikacji jest sprawdzić, w ilu przypadkach się mylimy, a w ilu model odpowiada poprawnie. Ta metryka jest zwana *__accuracy__*. Ma ona jednak zasadniczą wadę - kompletnie nie radzi sobie z klasami niezbalansowanymi.

Prosty przypadek - mamy zbiór danych, który pozwala na podstawie różnych parametrów medycznych wykryć rzadką chorobę, która zdarza się u 0.01% ludzi. Weźmy prosty klasyfikator, który zawsze zwraca klasę negatywną. Niby jest w oczywisty sposób kompletnie nieprzydatny, ale jednak dla losowej próbki ludzi dostanie *__accuracy__* równe 99.99%, bo, rzeczywiście, u większości tej choroby nie będzie.

Potrzebujemy bardziej skomplikowanej metryki, której nie da się tak łatwo oszukać.

Confusion Matrix

Żeby zdefiniować taką metrykę, musimy najpierw rozważyć jakie sytuacje mogą zdarzyć się przy klasyfikacji binarnej. Spójrzmy na tablicę poniżej:



- *__true positive__* - model zwrócił klasę pozytywną (*positive*), i jest to prawda (*true*)
- *__true negative__* - model zwrócił klasę negatywną (*negative*), i jest to prawda (*true*)
- *__false negative__* - model zwrócił klasę negatywną (*negative*), ale nie jest to prawda (*false*)
- *__false positive__* - model zwrócił klasę pozytywną (*positive*), ale nie jest to prawda (*false*)

Mając powyższe punkty - możemy zdefiniować *__accuracy__* następująco:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

czyli ilość przypadków, w których poprawnie zidentykowaliśmy klasę, podzieloną przez ilość wszystkich przypadków.

Precyzja i czułość (*Precision & Recall*)

Jednak jak zauważyliśmy wcześniej, istnieją sytuacje, w których nie jest to właściwe podejście.

Zdecydowanie ciekawszą dla nas metryką może być stwierdzenie jaką część rekordów z klasą pozytywną model poprawnie rozpoznał. Pozwoli to nam powiedzieć, jak czuły jest nasz model na klasę pozytywną. Ta metryka nazywa się czułością (***recall***):

$$recall = \frac{TP}{TP + FN}$$

czyli ilość przypadków, w których poprawnie rozpoznaliśmy klasę pozytywną, podzielona przez ilość wszystkich przypadków z klasą pozytywną.

Drugą korzystną dla nas metryką będzie stwierdzenie ile z osób, których zakwalifikowaliśmy klasą pozytywną, rzeczywiście do niej należy. Pozwoli to oszacować, jak często mylimy się oznaczając rekord klasą pozytywną. Ta metryka nazywa się precyzją (***precision***):

$$precision = \frac{TP}{TP + FP}$$

czyli ilość przypadków, w których poprawnie rozpoznaliśmy klasę pozytywną, podzielona przez ilość wszystkich przypadków, w których zwróciliśmy klasę pozytywną.

Ta metryka może być bardzo pomocna, na przykład, przy klasyfikacji spamu. Gorzej będzie, jeśli wrzucimy ważnego maila do spamu, niż przegapimy jakąś reklamę. Chcemy, aby jeśli coś zostało zaklasyfikowane jako spam, rzeczywiście nim było - chcemy jak najwyższą precyzję.

F1 score

Powyższe metryki mają wadę - pojedynczo ich łatwo oszukać:

- chcemy idealny ***precision***? - wystarczy zawsze zwracać klasę negatywną (ważny mail nie trafi do spamu, jeśli żadnego z nich tam nie wrzucimy)
- chcemy idealny ***recall***? - zawsze zwracamy klasę pozytywną (na pewno nie pominiemy chorego pacjenta, jeśli każdemu powiemy, że jest chory)

Musimy stosować ich w parze. Dla prostoty często agregujemy ich do jednej liczby za pomocą średniej harmonicznej. W przypadku liczb z zakresu $[0, 1]$ (a z takimi mamy sprawę), ona ma taką własność, że wartość wynikowa zawsze będzie bliżej do mniejszej liczby. I im większa jest między nimi różnica, tym bardziej jest to odczuwalne. Przykładowo, dla pary $(100\%, 0\%)$ średnia harmoniczna wynosi 0%. Średnia harmoniczna z ***precision*** i ***recall*** nazywa się ***f1 score***:

$$f1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Ten tutorial ma świetne wizualizację, które w interaktywny sposób prezentują działanie powyższych metryk.

Zadanie 10 (2.0 punkty)

Trening, tuning i analiza modeli

1. Wytrenuj podstawowy model regresji logistycznej z użyciem `LogisticRegression`. Użyj wag klas (`class_weights="balanced"`). Przetestuj model, wypisując precision, recall oraz f1-score w procentach. **Uwaga:** Scikit-learn domyślnie stosuje tutaj regularyzację L2, więc przekaż `penalty="None"`.
2. Dokonaj tuningu modelu z regularyzacją L2 za pomocą `LogisticRegressionCV`:
   - sprawdź 100 wartości, wystarczy podać liczbę do `Cs`
   - użyj 5-krotnej walidacji skrośnej
   - wybierz najlepszy model według metryki f1-score (parametr `scoring`)
   - pamiętaj o `class_weights="balanced"` i `random_state=0`
   - użyj `n_jobs=-1` dla przyspieszenia obliczeń (`-1` znaczy, że użyjemy wszystkich rdzeni do obliczeń)
   - przetestuj model, wypisując precision, recall oraz f1-score w procentach
   - **uwaga:** Scikit-learn stosuje tutaj konwencję, gdzie parametr `C` to odwrotność siły regularyzacji - im mniejszy, tym silniejsza regularyzacja.
3. Dokonaj analogicznego tuningu, ale dla regularyzacji L1. Użyj solwera SAGA. Przetestuj model, wypisując precision, recall oraz f1-score w procentach.
4. Dokonaj analizy wytrenowanych modeli:
   - oblicz f1-score na zbiorze treningowym modelu bez żadnej regularyzacji i porównaj go z wynikiem testowym; czy występuje tutaj overfitting?
   - czy twoim zdaniem tworzenie modeli z regularyzacją ma sens w tym przypadku?

Napisz co, w twojej opinii, jest ważniejsze dla naszego problemu, ***precision*** czy ***recall***? Jak moglibyśmy, nie zmieniając modelu, zmienić ich stosunek?

In [ ]:
```python
from sklearn.metrics import precision_score, recall_score, f1_score

def assess_logistic_regression_model(model, X_train, X_test, y_train, y_test):
    y_pred = model.predict(X_test)

    print(f"[PRECISSION]: {precision_score(y_test, y_pred):.4f}")
    print(f"[RECALL]:     {recall_score(y_test, y_pred):.4f}")
    print(f"[f1-score]:   {f1_score(y_test, y_pred, average='weighted'):.4f}%")
```

In [ ]:
```python
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV

reg_logistic = LogisticRegression(
    class_weight="balanced",
    penalty="l2",
    max_iter=10000
)

reg_logistic.fit(X_train, y_train)
assess_logistic_regression_model(reg_logistic, X_train, X_test, y_train, y_test)
```

```
[PRECISSION]: 0.2634
[RECALL]:     0.6704
[f1-score]:   0.7924%
```

In [ ]:
```python
reg_logistic_cv = LogisticRegressionCV(
    Cs=100,
    random_state=0,
    class_weight="balanced",
    n_jobs=-1,
    penalty="l2",
    cv=5,
    scoring="f1",
    max_iter=10000
).fit(X_train, y_train)

reg_logistic = LogisticRegression(
    class_weight="balanced",
    penalty="l2",
    max_iter=10000,
    C = reg_logistic_cv.C_[0],
).fit(X_train, y_train)

assess_logistic_regression_model(reg_logistic, X_train, X_test, y_train, y_test)
```

```
[PRECISSION]: 0.2634
[RECALL]:     0.6695
[f1-score]:   0.7925%
```

In [ ]:
```python
reg_logistic_cv = LogisticRegressionCV(
    penalty="l1",
    Cs=100,
    random_state=0,
    class_weight="balanced",
    n_jobs=-1,
    cv=5,
    scoring="f1",
    solver="saga",
    max_iter=10000,
    tol=0.0005
).fit(X_train, y_train)

reg_logistic = LogisticRegression(
    class_weight="balanced",
    penalty="l2",
    max_iter=10000,
    C = reg_logistic_cv.C_[0],
).fit(X_train, y_train)

assess_logistic_regression_model(reg_logistic, X_train, X_test, y_train, y_test)
```

```
[PRECISSION]: 0.2635
[RECALL]:     0.6695
[f1-score]:   0.7926%
```

W naszym przypadku zdecydowanie ważniejsza będzie metryka recall. Jest to spowodowane faktem, że koszty nierozpoznania osoby chętnej do założenia lokaty w banku `(FF)` będą znacznie większe od kosztów fałszywego rozpoznania osoby chętnej `(FP)`. Osoba która nie będzie chętna do założenia lokaty po prostu zignoruje nasz marketing, osoba do której nie dotrzemy nie będzie wiedziała że ma taką opcję.

Możemy zmienić stosunek tych metrych zmieniając wartość progu decyzyjnego. Domyślnie obserwacja przypisywana jest do klasy pozytywnej jeżeli jej prawdopodobieństwo należenia do tej klasy wynosi ponad `0.5`. Jeżeli uznamy że potrzebujemy zwiększyć metrykę **recall** możemy zmniejszyć wartość progu decyzyjnego. Jeżeli chcemy aby obserwacje były bardziej precyzyjne możemy ustawić bardziej restrykcyjny próg.

Zadanie 11 (2.0 punkty)

Dodanie cech wielomianowych do regresji logistycznej

1. Stwórz nowy pipeline do przetwarzania danych do regresji logistycznej, dodając `PolynomialFeatures` do zmiennych numerycznych przed standaryzacją. Wygeneruj cechy o stopniu 2, interakcje oraz potęgi, nie generuj interceptu.
2. Wytrenuj model regresji logistycznej bez regularyzacji na takim powiększonym zbiorze. Wypisz F1 treningowy oraz testowy w procentach.
3. Zdecyduj, czy jest sens tworzyć modele z regularyzacją. Jeżeli tak, to wytrenuj i dokonaj tuningu takich modeli. Jeżeli nie, to uzasadnij czemu.

In [ ]:
```
categorical_features = df.select_dtypes(include="object").columns
numerical_features = df.select_dtypes(exclude="object").columns

X_train, X_test, y_train, y_test = train_test_split(
    df, y, test_size=0.25, train_size=0.75, random_state=0, stratify=y
)
```

In [ ]:
```
poly_features = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)

categorical_pipeline = Pipeline(
    steps=[
        ('one_hot_encoder', one_hot_encoder)
    ]
)

numerical_pipeline = Pipeline(
    steps=[
        ('polynomial_features', poly_features),
        ('scaler', standard_scaler)
    ]
)

column_transformer = ColumnTransformer(
    transformers=[
        ("categorical_pipeline", categorical_pipeline, categorical_features),
        ("numerical_pipeline", numerical_pipeline, numerical_features)
    ],
    verbose_feature_names_out=False)

X_train_poly = column_transformer.fit_transform(X_train, y_train)
X_test_poly = column_transformer.transform(X_test)
```

In [ ]:
```
reg_logistic = LogisticRegression(
    penalty="l2",
    max_iter=10000,
    class_weight="balanced",
)

reg_logistic.fit(X_train_poly, y_train)
assess_logistic_regression_model(reg_logistic, X_train_poly, X_test_poly, y_train, y_test)
```

```
[PRECISSION]: 0.3057
[RECALL]:     0.6549
[f1-score]:   0.8231%
```

W pewnych sytuacjach warto zdecydować się na model z regularyzacją. Są to sytuacje gdy występuje nadmierne dopasowanie albo gdy model jest zbyt skomplikowany. W przypadku gdy nasz zbiór jest mniejszy a wszystkie cechy podobnie istotne regularyzacja może nie być optymalnym wyborem.

Zadanie dodatkowe (3 punkty)

Z formalnego, statystycznego punktu widzenia regresja liniowa czyni szereg założeń (Wikipedia):

1. Liniowość - relacja w danych może być reprezentowana jako `y=Xw`.
2. Normalność błędów - błędy (rezydua) mają rozkład normalny, wycentrowany na zerze.
3. Homoskedastyczność (stała wariancja) - wariancja błędu nie zależy od wartości docelowych `y`. Innymi słowy, nasz błąd będzie w przybliżeniu miał podobny "rozrzut" dla małych i dużych wartości `y`.

4. Niezależność błędów - błąd i `y` są niezależne (w sensie statystycznym). Innymi słowy, nie ma między nimi bezpośredniej relacji. Jeżeli nie pracujemy z szeregami czasowymi, to to założenie po prostu jest spełnione.
5. Brak współliniowości zmiennych - nie ma idealnej korelacji cech.

Testowanie tych własności nie zawsze jest oczywiste, a w szczególności Scikit-learn oferuje tutaj dość mało opcji, bo pochodzą one głównie z tradycyjnej statystyki.

1. Liniowość:

- numerycznie: wysoki współczynnik dopasowania modelu R^2 na zbiorze treningowym, niski błąd (RMSE) na zbiorze treningowym oraz testowym
- testem statystycznym: Rainbow test lub Harvey Collier test
- graficznie: możliwe kiedy mamy 1/2 zmienne i da się narysować wykres zmiennej zależnej względem cech

2. Normalność błędów:

- graficznie: robimy histogram rezyduów, powinien mieć kształt rozkładu normalnego i być wycentrowany na zerze
- testem statystycznym: Jarque-Bera test, Omnibus normality test

3. Homoskedastyczność:

- graficznie: robimy scatter plot rezyduów dla wartości przewidywanych od najmniejszej do największej, nie powinno być na nim żadnych widocznych wzorców czy kształtów; przykład 1, przykład 2
- testem statystycznym: Breusch–Pagan test lub Goldfeld-Quandt test

4. Niezależność błędów - nie omawiam, bo dotyczy tylko szeregów czasowych.
5. Brak współliniowości zmiennych: numerycznie, sprawdzić korelacje zmiennych, lub współczynnik uwarunkowania macierzy `X`

W ramach zadania wytrenuj model regresji liniowej dla zbioru danych Ames Housing z użyciem biblioteki Statsmodels: OLS docs, OLS, Regression diagnostics. Wytrenuj najpierw model bez regularyzacji, a następnie z regularyzacją L2 oraz L1. Nie przeprowadzaj tuningu, użyj tych wartości siły regularyzacji, które wyznaczyliśmy wcześniej.

Przetestuj założenia za pomocą testów statystycznych: Harvey Collier, Jarque-Bera, Breusch–Pagan. Współliniowość zmiennych zweryfikuj z użyciem współczynnika uwarunkowania. Zastosuj poziom istotności $\alpha = 0.05$.

Czy założenia są spełnione w przypadku podstawowego modelu i/lub modeli z regularyzacją? Czy modele regularyzowane w lepszym stopniu spełniają założenia?

In [ ]:
```
import statsmodels.api as sm

categorical_features = df_ames.select_dtypes(include="object").columns
numerical_features = df_ames.select_dtypes(exclude="object").columns

X_train, X_test, y_train, y_test = train_test_split(
    df_ames, y_ames, test_size=0.3, random_state=0
)
```

In [ ]:
```
df_ames.info
```

Out[ ]:
```
<bound method DataFrame.info of      MSSubClass MSZoning  LotFrontage  LotArea  Street  Alley  LotShape
\
0          SC20       RL        141.0    31770       1      0         3
1          SC20       RH         80.0    11622       1      0         4
2          SC20       RL         81.0    14267       1      0         3
3          SC20       RL         93.0    11160       1      0         4
4          SC60       RL         74.0    13830       1      0         3
...         ...      ...          ...      ...     ...    ...       ...
2925       SC80       RL         37.0     7937       1      0         3
2926       SC20       RL          0.0     8885       1      0         3
2927       SC85       RL         62.0    10441       1      0         4
2928       SC20       RL         77.0    10010       1      0         4
2929       SC60       RL         74.0     9627       1      0         4

     LandContour  Utilities LotConfig  ...  ScreenPorch PoolArea PoolQC  \
0            Lvl          4    Corner  ...            0        0      0
1            Lvl          4    Inside  ...          120        0      0
2            Lvl          4    Corner  ...            0        0      0
3            Lvl          4    Corner  ...            0        0      0
4            Lvl          4    Inside  ...            0        0      0
...          ...        ...       ...  ...          ...      ...    ...
2925         Lvl          4   CulDSac  ...            0        0      0
2926         Low          4    Inside  ...            0        0      0
2927         Lvl          4    Inside  ...            0        0      0
2928         Lvl          4    Inside  ...            0        0      0
2929         Lvl          4    Inside  ...            0        0      0

      Fence MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
0        No          No       0     May    2010       WD          Normal
1     MnPrv          No       0     Jun    2010       WD          Normal
2        No        Gar2   12500     Jun    2010       WD          Normal
3        No          No       0     Apr    2010       WD          Normal
4     MnPrv          No       0     Mar    2010       WD          Normal
...     ...         ...     ...     ...     ...       ...           ...
2925  GdPrv          No       0     Mar    2006       WD          Normal
2926  MnPrv          No       0     Jun    2006       WD          Normal
2927  MnPrv        Shed     700     Jul    2006       WD          Normal
2928     No          No       0     Apr    2006       WD          Normal
2929     No          No       0     Nov    2006       WD          Normal

[2922 rows x 79 columns]>
```

In [ ]:
```python
one_hot_encoder = OneHotEncoder(
    drop="first", sparse_output=False, handle_unknown="ignore"
)
median_imputer = SimpleImputer(strategy="median")
min_max_scaler = MinMaxScaler()

categorical_pipeline = Pipeline(
    steps=[
        ('one_hot_encoder', one_hot_encoder)
    ]
)

numerical_pipeline = Pipeline(
    steps=[
        ('imputer', median_imputer),
        ('scaler', min_max_scaler)
    ]
)

column_transformer = ColumnTransformer(
    transformers=[
        ("categorical_pipeline", categorical_pipeline, categorical_features),
        ("numerical_pipeline", numerical_pipeline, numerical_features)
    ],
    verbose_feature_names_out=False)

X_train = column_transformer.fit_transform(X_train, y_train)
X_test = column_transformer.transform(X_test)
```

```
/home/dominiq/anaconda3/lib/python3.10/site-packages/sklearn/preprocessing/_encoders.py:202: UserWarnin
g: Found unknown categories in columns [12, 15, 17] during transform. These unknown categories will be
encoded as all zeros
  warnings.warn(
```

In [ ]:
```python
def get_rmse(model, X_train, X_test, y_train, y_test) -> tuple:
    y_pred_train    = model.predict(X_train)
    y_pred_test     = model.predict(X_test)
```

```
    y_pred_test     = np.expm1(y_pred_test)
    y_pred_train    = np.expm1(y_pred_train)
    y_test          = np.expm1(y_test)
    y_train         = np.expm1(y_train)

    y_pred_test = [elem if not np.isinf(elem) else np.ma.masked_invalid(y_pred_test).mean() for elem in

    rmse_train  = mean_squared_error(y_train, y_pred_train, squared=False)
    rmse_test   = mean_squared_error(y_test, y_pred_test, squared=False)

    print(f"[TRAIN] RMSE: {rmse_train:.2f}")
    print(f"[TEST] RMSE: {rmse_test:.2f}")

    return (rmse_train, rmse_test)
```

In [ ]:

```
from statsmodels.stats.diagnostic import het_breuschpagan, linear_harvey_collier, linear_rainbow
from statsmodels.stats.stattools import jarque_bera

X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

results = sm.OLS(y_train, X_train).fit()
rmse_train_ols, rmse_test_ols = get_rmse(results, X_train, X_test, y_train, y_test)

het_bp_ols = het_breuschpagan(results.resid, X_train)
het_rainbow_ols = linear_rainbow(results)
het_jb_ols = jarque_bera(results.resid)

results_r_ols = results.rsquared
print(results.summary())
```

[TRAIN] RMSE: 16745.60
[TEST] RMSE: 21323.38

OLS Regression Results

Dep. Variable:	SalePrice	R-squared:	0.940
Model:	OLS	Adj. R-squared:	0.933
Method:	Least Squares	F-statistic:	124.4
Date:	Tue, 17 Oct 2023	Prob (F-statistic):	0.00
Time:	10:39:02	Log-Likelihood:	1816.4
No. Observations:	2045	AIC:	-3171.
Df Residuals:	1814	BIC:	-1872.
Df Model:	230		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	8.3033	0.284	29.239	0.000	7.746	8.860
x1	-0.1753	0.124	-1.418	0.156	-0.418	0.067
x2	-0.0635	0.035	-1.797	0.073	-0.133	0.006
x3	-0.0173	0.055	-0.316	0.752	-0.125	0.090
x4	0.1460	0.110	1.332	0.183	-0.069	0.361
x5	0.0730	0.051	1.428	0.154	-0.027	0.173
x6	0.0159	0.054	0.295	0.768	-0.090	0.121
x7	0.0539	0.078	0.695	0.487	-0.098	0.206
x8	0.0185	0.090	0.206	0.837	-0.158	0.195
x9	0.1356	0.061	2.223	0.026	0.016	0.255
x10	0.0549	0.058	0.950	0.342	-0.058	0.168
x11	0.1183	0.059	2.005	0.045	0.003	0.234
x12	0.0820	0.075	1.091	0.275	-0.065	0.229
x13	0.0055	0.077	0.072	0.943	-0.145	0.156
x14	0.1067	0.065	1.643	0.101	-0.021	0.234
x15	0.0095	0.030	0.318	0.750	-0.049	0.068
x16	0.9676	0.157	6.179	0.000	0.660	1.275
x17	1.2416	0.157	7.924	0.000	0.934	1.549
x18	1.1919	0.178	6.701	0.000	0.843	1.541
x19	1.1959	0.157	7.593	0.000	0.887	1.505
x20	1.2304	0.154	7.965	0.000	0.927	1.533
x21	1.1752	0.154	7.637	0.000	0.873	1.477
x22	0.0212	0.020	1.056	0.291	-0.018	0.060
x23	0.0038	0.025	0.154	0.877	-0.045	0.052
x24	0.0063	0.014	0.442	0.658	-0.022	0.034
x25	0.0058	0.013	0.458	0.647	-0.019	0.031
x26	-0.0254	0.017	-1.474	0.141	-0.059	0.008
x27	-0.0193	0.034	-0.566	0.572	-0.086	0.048
x28	-0.0040	0.007	-0.581	0.562	-0.017	0.009
x29	0.1238	0.061	2.026	0.043	0.004	0.244
x30	0.0529	0.044	1.192	0.233	-0.034	0.140
x31	0.0831	0.036	2.300	0.022	0.012	0.154
x32	0.0613	0.037	1.651	0.099	-0.012	0.134
x33	0.0122	0.029	0.424	0.671	-0.044	0.068
x34	0.1167	0.033	3.508	0.000	0.051	0.182
x35	-0.0133	0.031	-0.427	0.670	-0.074	0.048
x36	0.0137	0.030	0.451	0.652	-0.046	0.073
x37	0.0973	0.052	1.860	0.063	-0.005	0.200
x38	0.0244	0.040	0.616	0.538	-0.053	0.102
x39	-0.0696	0.045	-1.533	0.125	-0.159	0.019
x40	0.0193	0.031	0.619	0.536	-0.042	0.080
x41	0.0166	0.031	0.543	0.587	-0.043	0.077
x42	0.0821	0.053	1.546	0.122	-0.022	0.186
x43	0.0062	0.031	0.198	0.843	-0.055	0.068
x44	0.0651	0.034	1.899	0.058	-0.002	0.132
x45	0.0923	0.029	3.164	0.002	0.035	0.150
x46	0.0107	0.037	0.292	0.770	-0.061	0.082
x47	0.0313	0.039	0.806	0.420	-0.045	0.107
x48	0.0290	0.032	0.910	0.363	-0.033	0.091
x49	0.0082	0.030	0.270	0.787	-0.051	0.068
x50	0.0732	0.035	2.112	0.035	0.005	0.141
x51	0.1194	0.033	3.598	0.000	0.054	0.184
x52	0.0308	0.032	0.972	0.331	-0.031	0.093
x53	-0.0033	0.040	-0.083	0.934	-0.081	0.075
x54	0.0138	0.019	0.733	0.464	-0.023	0.051
x55	0.0573	0.016	3.692	0.000	0.027	0.088
x56	0.0786	0.035	2.245	0.025	0.010	0.147
x57	0.0774	0.027	2.831	0.005	0.024	0.131
x58	0.0044	0.030	0.147	0.883	-0.054	0.062
x59	0.0161	0.029	0.565	0.572	-0.040	0.072
x60	0.0033	0.066	0.050	0.960	-0.126	0.133
x61	0.0068	0.047	0.145	0.885	-0.085	0.098
x62	-0.0788	0.072	-1.088	0.277	-0.221	0.063
x63	-0.0352	0.061	-0.577	0.564	-0.155	0.084

x64	0.0330	0.087	0.381	0.703	-0.137	0.203
x65	-0.0218	0.101	-0.215	0.830	-0.220	0.177
x66	0.0140	0.185	0.075	0.940	-0.350	0.377
x67	-0.0520	0.126	-0.414	0.679	-0.299	0.195
x68	0.0344	0.125	0.274	0.784	-0.212	0.280
x69	-0.0648	0.099	-0.653	0.514	-0.259	0.130
x70	0.0095	0.030	0.318	0.750	-0.049	0.068
x71	-0.0244	0.055	-0.447	0.655	-0.131	0.083
x72	0.0218	0.051	0.425	0.671	-0.079	0.122
x73	0.1197	0.073	1.636	0.102	-0.024	0.263
x74	0.0583	0.034	1.720	0.086	-0.008	0.125
x75	-0.0264	0.068	-0.387	0.699	-0.160	0.108
x76	0.1102	0.051	2.140	0.032	0.009	0.211
x77	0.0545	0.032	1.725	0.085	-0.007	0.116
x78	0.0388	0.046	0.836	0.403	-0.052	0.130
x79	0.1091	0.062	1.761	0.078	-0.012	0.231
x80	0.0203	0.062	0.328	0.743	-0.101	0.141
x81	0.0123	0.068	0.181	0.857	-0.122	0.146
x82	0.0223	0.062	0.359	0.720	-0.099	0.144
x83	-0.1176	0.078	-1.514	0.130	-0.270	0.035
x84	-0.0937	0.127	-0.737	0.461	-0.343	0.156
x85	0.1833	0.126	1.449	0.147	-0.065	0.431
x86	0.0847	0.127	0.667	0.505	-0.164	0.334
x87	0.0890	0.114	0.778	0.437	-0.135	0.313
x88	-0.0340	0.056	-0.608	0.543	-0.144	0.076
x89	-0.0060	0.051	-0.118	0.906	-0.107	0.095
x90	0.1729	0.068	2.527	0.012	0.039	0.307
x91	-0.0913	0.123	-0.743	0.458	-0.332	0.150
x92	0.0643	0.073	0.885	0.376	-0.078	0.207
x93	0.1384	0.047	2.955	0.003	0.047	0.230
x94	1.2693	0.183	6.919	0.000	0.909	1.629
x95	0.0055	0.090	0.061	0.951	-0.171	0.182
x96	0.0464	0.045	1.030	0.303	-0.042	0.135
x97	-0.0107	0.122	-0.088	0.930	-0.249	0.228
x98	0.0148	0.056	0.263	0.792	-0.096	0.125
x99	0.0523	0.044	1.179	0.239	-0.035	0.139
x100	0.3068	0.060	5.143	0.000	0.190	0.424
x101	-0.0275	0.101	-0.272	0.786	-0.226	0.171
x102	0.0551	0.052	1.052	0.293	-0.048	0.158
x103	0.0379	0.050	0.752	0.452	-0.061	0.137
x104	0.0587	0.044	1.330	0.184	-0.028	0.145
x105	0.0720	0.048	1.491	0.136	-0.023	0.167
x106	0.1542	0.095	1.627	0.104	-0.032	0.340
x107	-0.0244	0.067	-0.367	0.714	-0.155	0.106
x108	-0.0168	0.050	-0.336	0.737	-0.115	0.081
x109	0.0824	0.090	0.911	0.362	-0.095	0.260
x110	0.0123	0.045	0.273	0.785	-0.076	0.101
x111	0.0184	0.054	0.340	0.734	-0.088	0.125
x112	0.0663	0.056	1.175	0.240	-0.044	0.177
x113	-0.0547	0.121	-0.454	0.650	-0.291	0.182
x114	0.0024	0.043	0.054	0.957	-0.083	0.088
x115	0.3068	0.060	5.143	0.000	0.190	0.424
x116	0.0814	0.079	1.029	0.304	-0.074	0.237
x117	0.0439	0.052	0.844	0.399	-0.058	0.146
x118	0.0267	0.050	0.530	0.596	-0.072	0.125
x119	0.0132	0.044	0.299	0.765	-0.073	0.100
x120	0.0004	0.047	0.009	0.993	-0.092	0.093
x121	0.0317	0.027	1.185	0.236	-0.021	0.084
x122	-0.1895	0.160	-1.185	0.236	-0.503	0.124
x123	0.0377	0.027	1.405	0.160	-0.015	0.090
x124	0.0545	0.028	1.927	0.054	-0.001	0.110
x125	0.0187	0.012	1.576	0.115	-0.005	0.042
x126	0.0352	0.013	2.697	0.007	0.010	0.061
x127	0.0216	0.033	0.648	0.517	-0.044	0.087
x128	0.0713	0.053	1.334	0.182	-0.033	0.176
x129	-0.0696	0.053	-1.319	0.187	-0.173	0.034
x130	0.0891	0.113	0.789	0.430	-0.132	0.311
x131	0.1314	0.116	1.136	0.256	-0.095	0.358
x132	-0.0247	0.121	-0.205	0.838	-0.262	0.212
x133	0.0689	0.143	0.482	0.630	-0.211	0.349
x134	0.0378	0.014	2.752	0.006	0.011	0.065
x135	-0.0037	0.022	-0.165	0.869	-0.048	0.040
x136	0.0504	0.054	0.941	0.347	-0.055	0.155
x137	-0.0080	0.012	-0.693	0.488	-0.031	0.015
x138	0.0503	0.110	0.457	0.648	-0.166	0.266
x139	0.0510	0.030	1.695	0.090	-0.008	0.110
x140	0.0640	0.041	1.576	0.115	-0.016	0.144
x141	0.0502	0.033	1.538	0.124	-0.014	0.114
x142	-0.0036	0.045	-0.081	0.935	-0.092	0.085
x143	0.0342	0.030	1.142	0.254	-0.025	0.093

x144	0.0380	0.101	0.375	0.707	-0.160	0.236
x145	0.0761	0.108	0.704	0.482	-0.136	0.288
x146	-0.0118	0.008	-1.564	0.118	-0.027	0.003
x147	-0.0007	0.009	-0.079	0.937	-0.018	0.017
x148	-0.0154	0.018	-0.860	0.390	-0.050	0.020
x149	0.0095	0.015	0.650	0.516	-0.019	0.038
x150	-0.0268	0.047	-0.575	0.566	-0.118	0.065
x151	0.0056	0.013	0.427	0.669	-0.020	0.031
x152	0.0440	0.101	0.434	0.664	-0.155	0.243
x153	0.0763	0.111	0.690	0.491	-0.141	0.293
x154	0.0327	0.095	0.344	0.731	-0.154	0.219
x155	-0.1992	0.155	-1.283	0.200	-0.504	0.105
x156	-0.0103	0.012	-0.844	0.399	-0.034	0.014
x157	-0.0004	0.015	-0.024	0.981	-0.030	0.030
x158	-0.0050	0.014	-0.356	0.722	-0.032	0.022
x159	0.0106	0.014	0.743	0.457	-0.017	0.039
x160	0.0027	0.010	0.267	0.789	-0.017	0.023
x161	-0.0083	0.010	-0.821	0.412	-0.028	0.012
x162	-0.0036	0.012	-0.291	0.771	-0.028	0.021
x163	0.0074	0.011	0.698	0.485	-0.013	0.028
x164	-0.0211	0.014	-1.505	0.132	-0.049	0.006
x165	-0.0097	0.013	-0.735	0.463	-0.036	0.016
x166	-2.847e-05	0.013	-0.002	0.998	-0.026	0.026
x167	0.0033	0.038	0.087	0.931	-0.072	0.078
x168	0.1240	0.052	2.374	0.018	0.022	0.226
x169	0.0705	0.031	2.268	0.023	0.010	0.131
x170	-0.0406	0.045	-0.911	0.362	-0.128	0.047
x171	0.0070	0.059	0.117	0.907	-0.110	0.124
x172	0.0873	0.055	1.600	0.110	-0.020	0.194
x173	0.1484	0.051	2.911	0.004	0.048	0.248
x174	-0.0028	0.111	-0.025	0.980	-0.220	0.214
x175	0.0034	0.016	0.221	0.825	-0.027	0.034
x176	0.2201	0.047	4.710	0.000	0.128	0.312
x177	0.1235	0.038	3.238	0.001	0.049	0.198
x178	0.0472	0.022	2.123	0.034	0.004	0.091
x179	0.1021	0.011	9.201	0.000	0.080	0.124
x180	0.0515	0.052	0.990	0.322	-0.051	0.154
x181	0.0124	0.028	0.449	0.653	-0.042	0.067
x182	0.4771	0.089	5.367	0.000	0.303	0.651
x183	0.0084	0.048	0.175	0.861	-0.086	0.103
x184	-0.0076	0.016	-0.464	0.642	-0.040	0.024
x185	-0.0010	0.016	-0.063	0.950	-0.033	0.031
x186	0.1692	0.121	1.395	0.163	-0.069	0.407
x187	-0.0206	0.027	-0.776	0.438	-0.073	0.031
x188	0.4770	0.035	13.531	0.000	0.408	0.546
x189	0.3259	0.027	12.257	0.000	0.274	0.378
x190	0.2530	0.043	5.877	0.000	0.169	0.337
x191	0.0291	0.013	2.282	0.023	0.004	0.054
x192	0.0525	0.036	1.456	0.146	-0.018	0.123
x193	0.0247	0.024	1.014	0.311	-0.023	0.073
x194	0.0245	0.031	0.797	0.425	-0.036	0.085
x195	0.0564	0.032	1.781	0.075	-0.006	0.119
x196	0.0394	0.039	1.013	0.311	-0.037	0.116
x197	0.0370	0.010	3.736	0.000	0.018	0.056
x198	0.0161	0.012	1.346	0.179	-0.007	0.040
x199	0.1709	0.023	7.559	0.000	0.127	0.215
x200	0.0025	0.027	0.093	0.926	-0.051	0.056
x201	0.0702	0.035	2.003	0.045	0.001	0.139
x202	0.0054	0.020	0.277	0.782	-0.033	0.044
x203	0.1594	0.028	5.664	0.000	0.104	0.215
x204	0.0589	0.015	4.035	0.000	0.030	0.088
x205	0.4845	0.037	13.199	0.000	0.413	0.557
x206	0.1832	0.035	5.214	0.000	0.114	0.252
x207	0.1197	0.062	1.934	0.053	-0.002	0.241
x208	0.5416	0.033	16.372	0.000	0.477	0.606
x209	0.0439	0.022	2.006	0.045	0.001	0.087
x210	0.0053	0.022	0.238	0.812	-0.038	0.049
x211	0.0793	0.033	2.380	0.017	0.014	0.145
x212	0.0422	0.016	2.598	0.009	0.010	0.074
x213	-0.0731	0.041	-1.765	0.078	-0.154	0.008
x214	-0.0559	0.057	-0.989	0.323	-0.167	0.055
x215	0.0587	0.024	2.397	0.017	0.011	0.107
x216	0.0352	0.039	0.900	0.368	-0.042	0.112
x217	0.2074	0.031	6.771	0.000	0.147	0.267
x218	0.0684	0.021	3.309	0.001	0.028	0.109
x219	-0.0064	0.020	-0.319	0.750	-0.046	0.033
x220	0.0576	0.026	2.218	0.027	0.007	0.109
x221	0.1327	0.034	3.859	0.000	0.065	0.200
x222	0.0316	0.044	0.721	0.471	-0.054	0.118
x223	0.0157	0.068	0.231	0.817	-0.118	0.149

```
x224           0.1405      0.073      1.928      0.054      -0.002       0.283
x225           0.0333      0.013      2.626      0.009       0.008       0.058
x226           0.0651      0.032      2.058      0.040       0.003       0.127
x227           0.0246      0.032      0.768      0.442      -0.038       0.087
x228           0.1777      0.045      3.991      0.000       0.090       0.265
x229           0.0715      0.048      1.477      0.140      -0.023       0.167
x230           0.1365      0.026      5.202      0.000       0.085       0.188
x231           0.1427      0.160      0.890      0.373      -0.172       0.457
x232          -0.1205      0.177     -0.680      0.497      -0.468       0.227
x233           0.1307      0.148      0.881      0.378      -0.160       0.422
x234          -0.0163      0.008     -2.085      0.037      -0.032      -0.001
==============================================================================
Omnibus:                     1011.491   Durbin-Watson:                   2.005
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            34985.667
Skew:                          -1.684   Prob(JB):                         0.00
Kurtosis:                      22.981   Cond. No.                     1.64e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.03e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [ ]:

```python
from statsmodels.tools.tools import pinv_extended

model = sm.OLS(y_train, X_train)
results = model.fit_regularized(method="elastic_net", alpha=reg_lasso_.alpha_, L1_wt=0.0)

pinv_wexog,_ = pinv_extended(model.wexog)
normalized_cov_params = np.dot(pinv_wexog, np.transpose(pinv_wexog))
final = sm.regression.linear_model.OLSResults(model, results.params, normalized_cov_params)

rmse_train_lasso, rmse_test_lasso = get_rmse(results, X_train, X_test, y_train, y_test)
het_bp_lasso = het_breuschpagan(final.resid, X_train)
het_jb_lasso = jarque_bera(final.resid)

results_r_lasso = final.rsquared
print(final.summary())
```

[TRAIN] RMSE: 18507.51
[TEST] RMSE: 21326.87

OLS Regression Results

Dep. Variable:	SalePrice	R-squared:	0.920
Model:	OLS	Adj. R-squared:	0.910
Method:	Least Squares	F-statistic:	90.72
Date:	Tue, 17 Oct 2023	Prob (F-statistic):	0.00
Time:	10:39:02	Log-Likelihood:	1515.6
No. Observations:	2045	AIC:	-2569.
Df Residuals:	1814	BIC:	-1270.
Df Model:	230		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	3.7227	0.329	11.316	0.000	3.077	4.368
x1	-0.1462	0.143	-1.021	0.308	-0.427	0.135
x2	-0.0421	0.041	-1.029	0.304	-0.122	0.038
x3	-0.1708	0.064	-2.684	0.007	-0.296	-0.046
x4	0.1782	0.127	1.404	0.161	-0.071	0.427
x5	0.2230	0.059	3.763	0.000	0.107	0.339
x6	0.1899	0.062	3.047	0.002	0.068	0.312
x7	0.2426	0.090	2.699	0.007	0.066	0.419
x8	0.1660	0.104	1.594	0.111	-0.038	0.370
x9	0.4043	0.071	5.724	0.000	0.266	0.543
x10	0.2283	0.067	3.411	0.001	0.097	0.360
x11	0.3194	0.068	4.672	0.000	0.185	0.453
x12	0.2675	0.087	3.072	0.002	0.097	0.438
x13	-0.0583	0.089	-0.655	0.512	-0.233	0.116
x14	0.1727	0.075	2.295	0.022	0.025	0.320
x15	0.0033	0.035	0.096	0.923	-0.064	0.071
x16	0.3380	0.181	1.863	0.063	-0.018	0.694
x17	0.6291	0.182	3.466	0.001	0.273	0.985
x18	0.7615	0.206	3.696	0.000	0.357	1.166
x19	0.5722	0.182	3.136	0.002	0.214	0.930
x20	0.6356	0.179	3.552	0.000	0.285	0.987
x21	0.5575	0.178	3.128	0.002	0.208	0.907
x22	0.0405	0.023	1.746	0.081	-0.005	0.086
x23	0.0546	0.029	1.901	0.057	-0.002	0.111
x24	0.0195	0.017	1.180	0.238	-0.013	0.052
x25	0.0276	0.015	1.867	0.062	-0.001	0.057
x26	-0.0097	0.020	-0.487	0.626	-0.049	0.029
x27	-0.0148	0.039	-0.374	0.708	-0.092	0.063
x28	-0.0001	0.008	-0.018	0.986	-0.016	0.015
x29	0.1660	0.071	2.345	0.019	0.027	0.305
x30	0.1020	0.051	1.986	0.047	0.001	0.203
x31	0.1879	0.042	4.487	0.000	0.106	0.270
x32	0.1664	0.043	3.868	0.000	0.082	0.251
x33	0.0474	0.033	1.427	0.154	-0.018	0.112
x34	0.1904	0.039	4.938	0.000	0.115	0.266
x35	0.0561	0.036	1.559	0.119	-0.014	0.127
x36	0.0666	0.035	1.899	0.058	-0.002	0.135
x37	0.1052	0.061	1.736	0.083	-0.014	0.224
x38	0.1573	0.046	3.425	0.001	0.067	0.247
x39	-0.0138	0.053	-0.262	0.793	-0.117	0.089
x40	0.0821	0.036	2.276	0.023	0.011	0.153
x41	0.0684	0.035	1.931	0.054	-0.001	0.138
x42	0.0878	0.062	1.427	0.154	-0.033	0.208
x43	0.0405	0.036	1.116	0.264	-0.031	0.112
x44	0.0970	0.040	2.442	0.015	0.019	0.175
x45	0.1195	0.034	3.534	0.000	0.053	0.186
x46	0.1156	0.042	2.732	0.006	0.033	0.199
x47	0.0944	0.045	2.099	0.036	0.006	0.183
x48	0.0790	0.037	2.141	0.032	0.007	0.151
x49	0.0390	0.035	1.112	0.266	-0.030	0.108
x50	0.1117	0.040	2.781	0.005	0.033	0.190
x51	0.1509	0.038	3.926	0.000	0.076	0.226
x52	0.0962	0.037	2.618	0.009	0.024	0.168
x53	0.0114	0.046	0.247	0.805	-0.079	0.102
x54	0.0525	0.022	2.404	0.016	0.010	0.095
x55	0.0887	0.018	4.932	0.000	0.053	0.124
x56	0.1253	0.041	3.087	0.002	0.046	0.205
x57	0.1029	0.032	3.248	0.001	0.041	0.165
x58	0.0221	0.034	0.645	0.519	-0.045	0.089
x59	0.0560	0.033	1.695	0.090	-0.009	0.121
x60	0.0653	0.076	0.854	0.393	-0.085	0.215
x61	0.1298	0.054	2.397	0.017	0.024	0.236
x62	0.2715	0.084	3.235	0.001	0.107	0.436
x63	0.3424	0.071	4.848	0.000	0.204	0.481

x64	0.3369	0.100	3.353	0.001	0.140	0.534
x65	0.2748	0.117	2.343	0.019	0.045	0.505
x66	0.4076	0.215	1.898	0.058	-0.014	0.829
x67	0.2086	0.146	1.431	0.152	-0.077	0.494
x68	0.2254	0.145	1.551	0.121	-0.060	0.510
x69	0.0363	0.115	0.316	0.752	-0.189	0.262
x70	0.0033	0.035	0.096	0.923	-0.064	0.071
x71	0.1420	0.063	2.245	0.025	0.018	0.266
x72	0.1871	0.059	3.153	0.002	0.071	0.304
x73	0.2555	0.085	3.013	0.003	0.089	0.422
x74	0.1784	0.039	4.540	0.000	0.101	0.256
x75	0.0463	0.079	0.585	0.559	-0.109	0.201
x76	0.1478	0.060	2.479	0.013	0.031	0.265
x77	0.1349	0.037	3.687	0.000	0.063	0.207
x78	0.2102	0.054	3.910	0.000	0.105	0.316
x79	0.4314	0.072	6.009	0.000	0.291	0.572
x80	0.3311	0.072	4.629	0.000	0.191	0.471
x81	0.2932	0.079	3.709	0.000	0.138	0.448
x82	0.3288	0.072	4.571	0.000	0.188	0.470
x83	0.1656	0.090	1.841	0.066	-0.011	0.342
x84	0.2493	0.147	1.694	0.091	-0.039	0.538
x85	0.3234	0.147	2.207	0.027	0.036	0.611
x86	0.2330	0.147	1.583	0.114	-0.056	0.522
x87	0.1062	0.132	0.802	0.423	-0.154	0.366
x88	0.1863	0.065	2.878	0.004	0.059	0.313
x89	0.0027	0.059	0.046	0.964	-0.114	0.119
x90	0.1357	0.079	1.712	0.087	-0.020	0.291
x91	-0.1083	0.142	-0.761	0.447	-0.387	0.171
x92	0.1097	0.084	1.304	0.192	-0.055	0.275
x93	0.1534	0.054	2.826	0.005	0.047	0.260
x94	0.4803	0.213	2.260	0.024	0.063	0.897
x95	0.0772	0.104	0.740	0.460	-0.127	0.282
x96	0.0483	0.052	0.925	0.355	-0.054	0.151
x97	0.0280	0.141	0.199	0.842	-0.248	0.304
x98	0.0331	0.065	0.508	0.611	-0.095	0.161
x99	0.0586	0.051	1.140	0.254	-0.042	0.159
x100	0.2781	0.069	4.025	0.000	0.143	0.414
x101	-0.0373	0.117	-0.319	0.750	-0.267	0.192
x102	0.0523	0.061	0.862	0.389	-0.067	0.171
x103	0.0238	0.058	0.407	0.684	-0.091	0.138
x104	0.0691	0.051	1.351	0.177	-0.031	0.169
x105	0.0794	0.056	1.419	0.156	-0.030	0.189
x106	0.1352	0.110	1.232	0.218	-0.080	0.350
x107	0.0329	0.077	0.427	0.670	-0.118	0.184
x108	0.0282	0.058	0.488	0.626	-0.085	0.141
x109	0.0644	0.105	0.615	0.539	-0.141	0.270
x110	0.0729	0.052	1.398	0.162	-0.029	0.175
x111	0.0594	0.063	0.946	0.344	-0.064	0.183
x112	0.1086	0.065	1.660	0.097	-0.020	0.237
x113	0.0193	0.140	0.138	0.890	-0.255	0.293
x114	0.0516	0.050	1.025	0.305	-0.047	0.150
x115	0.2781	0.069	4.025	0.000	0.143	0.414
x116	0.1634	0.092	1.782	0.075	-0.016	0.343
x117	0.1075	0.060	1.783	0.075	-0.011	0.226
x118	0.0901	0.058	1.547	0.122	-0.024	0.204
x119	0.0618	0.051	1.207	0.228	-0.039	0.162
x120	0.0500	0.054	0.920	0.358	-0.057	0.157
x121	0.1314	0.031	4.243	0.000	0.071	0.192
x122	-0.0515	0.185	-0.278	0.781	-0.415	0.312
x123	0.1520	0.031	4.894	0.000	0.091	0.213
x124	0.1596	0.033	4.875	0.000	0.095	0.224
x125	0.0406	0.014	2.957	0.003	0.014	0.068
x126	0.0448	0.015	2.957	0.003	0.015	0.074
x127	0.0438	0.039	1.133	0.257	-0.032	0.119
x128	0.1096	0.062	1.771	0.077	-0.012	0.231
x129	-0.0410	0.061	-0.671	0.502	-0.161	0.079
x130	0.7651	0.131	5.849	0.000	0.509	1.022
x131	0.7826	0.134	5.842	0.000	0.520	1.045
x132	0.6315	0.140	4.509	0.000	0.357	0.906
x133	0.4631	0.166	2.796	0.005	0.138	0.788
x134	0.0217	0.016	1.363	0.173	-0.010	0.053
x135	0.0012	0.026	0.045	0.964	-0.050	0.052
x136	0.0681	0.062	1.098	0.272	-0.054	0.190
x137	0.0042	0.013	0.314	0.754	-0.022	0.030
x138	0.0225	0.127	0.176	0.860	-0.228	0.273
x139	0.1800	0.035	5.168	0.000	0.112	0.248
x140	0.2534	0.047	5.387	0.000	0.161	0.346
x141	0.1702	0.038	4.497	0.000	0.096	0.244
x142	0.1872	0.052	3.592	0.000	0.085	0.289
x143	0.1660	0.035	4.783	0.000	0.098	0.234

x144	0.2447	0.117	2.089	0.037	0.015	0.474
x145	0.3011	0.125	2.404	0.016	0.055	0.547
x146	0.0046	0.009	0.527	0.598	-0.013	0.022
x147	0.0185	0.011	1.761	0.078	-0.002	0.039
x148	0.0077	0.021	0.371	0.711	-0.033	0.048
x149	0.0360	0.017	2.129	0.033	0.003	0.069
x150	0.0187	0.054	0.346	0.729	-0.087	0.125
x151	0.0389	0.015	2.555	0.011	0.009	0.069
x152	0.6663	0.118	5.668	0.000	0.436	0.897
x153	0.3736	0.128	2.915	0.004	0.122	0.625
x154	0.6073	0.110	5.510	0.000	0.391	0.823
x155	0.1850	0.180	1.028	0.304	-0.168	0.538
x156	0.0046	0.014	0.326	0.744	-0.023	0.032
x157	0.0178	0.018	1.003	0.316	-0.017	0.053
x158	0.0108	0.016	0.670	0.503	-0.021	0.043
x159	0.0407	0.017	2.457	0.014	0.008	0.073
x160	0.0170	0.012	1.432	0.152	-0.006	0.040
x161	0.0011	0.012	0.093	0.926	-0.022	0.024
x162	0.0215	0.014	1.510	0.131	-0.006	0.049
x163	0.0225	0.012	1.842	0.066	-0.001	0.047
x164	-0.0062	0.016	-0.384	0.701	-0.038	0.026
x165	0.0041	0.015	0.264	0.792	-0.026	0.034
x166	0.0137	0.016	0.881	0.378	-0.017	0.044
x167	0.0082	0.044	0.184	0.854	-0.079	0.095
x168	0.1005	0.060	1.661	0.097	-0.018	0.219
x169	0.1653	0.036	4.590	0.000	0.095	0.236
x170	0.0013	0.052	0.026	0.979	-0.100	0.102
x171	0.0446	0.069	0.647	0.518	-0.091	0.180
x172	0.0876	0.063	1.387	0.166	-0.036	0.212
x173	0.1242	0.059	2.103	0.036	0.008	0.240
x174	-0.0028	0.128	-0.022	0.982	-0.254	0.249
x175	0.0265	0.018	1.471	0.141	-0.009	0.062
x176	0.2099	0.054	3.877	0.000	0.104	0.316
x177	0.1403	0.044	3.174	0.002	0.054	0.227
x178	0.0502	0.026	1.947	0.052	-0.000	0.101
x179	0.1048	0.013	8.157	0.000	0.080	0.130
x180	0.0699	0.060	1.159	0.247	-0.048	0.188
x181	0.0318	0.032	0.993	0.321	-0.031	0.095
x182	0.4895	0.103	4.753	0.000	0.288	0.691
x183	0.2440	0.056	4.386	0.000	0.135	0.353
x184	-0.0111	0.019	-0.588	0.556	-0.048	0.026
x185	0.0356	0.019	1.883	0.060	-0.001	0.073
x186	1.5420	0.141	10.971	0.000	1.266	1.818
x187	0.0775	0.031	2.518	0.012	0.017	0.138
x188	0.4479	0.041	10.968	0.000	0.368	0.528
x189	0.3322	0.031	10.785	0.000	0.272	0.393
x190	0.3436	0.050	6.891	0.000	0.246	0.441
x191	0.0036	0.015	0.242	0.809	-0.025	0.033
x192	0.0895	0.042	2.142	0.032	0.008	0.172
x193	0.0222	0.028	0.786	0.432	-0.033	0.078
x194	0.0410	0.036	1.151	0.250	-0.029	0.111
x195	0.0830	0.037	2.262	0.024	0.011	0.155
x196	0.0393	0.045	0.872	0.383	-0.049	0.128
x197	0.0512	0.011	4.467	0.000	0.029	0.074
x198	0.0181	0.014	1.305	0.192	-0.009	0.045
x199	0.1418	0.026	5.414	0.000	0.090	0.193
x200	0.0152	0.031	0.485	0.628	-0.046	0.077
x201	0.0692	0.041	1.704	0.089	-0.010	0.149
x202	-0.0072	0.023	-0.317	0.752	-0.052	0.037
x203	0.1289	0.033	3.954	0.000	0.065	0.193
x204	0.0738	0.017	4.363	0.000	0.041	0.107
x205	0.4568	0.043	10.741	0.000	0.373	0.540
x206	0.1711	0.041	4.204	0.000	0.091	0.251
x207	0.0914	0.072	1.275	0.202	-0.049	0.232
x208	0.5451	0.038	14.226	0.000	0.470	0.620
x209	0.0544	0.025	2.149	0.032	0.005	0.104
x210	0.0346	0.026	1.340	0.180	-0.016	0.085
x211	0.1052	0.039	2.725	0.006	0.029	0.181
x212	0.0527	0.019	2.804	0.005	0.016	0.090
x213	-0.0263	0.048	-0.547	0.584	-0.120	0.068
x214	0.2967	0.065	4.532	0.000	0.168	0.425
x215	0.0984	0.028	3.471	0.001	0.043	0.154
x216	0.0424	0.045	0.937	0.349	-0.046	0.131
x217	0.3242	0.035	9.138	0.000	0.255	0.394
x218	0.0766	0.024	3.198	0.001	0.030	0.124
x219	0.0279	0.023	1.198	0.231	-0.018	0.073
x220	0.0283	0.030	0.942	0.346	-0.031	0.087
x221	0.1793	0.040	4.501	0.000	0.101	0.257
x222	0.0310	0.051	0.610	0.542	-0.069	0.130
x223	0.2512	0.079	3.192	0.001	0.097	0.406

```
x224           0.3549      0.084      4.204      0.000       0.189       0.520
x225           0.0349      0.015      2.373      0.018       0.006       0.064
x226           0.0747      0.037      2.039      0.042       0.003       0.147
x227           0.0552      0.037      1.490      0.136      -0.017       0.128
x228           0.1648      0.052      3.194      0.001       0.064       0.266
x229           0.0525      0.056      0.936      0.350      -0.058       0.163
x230           0.1634      0.030      5.376      0.000       0.104       0.223
x231           0.0997      0.186      0.537      0.591      -0.264       0.464
x232          -0.0159      0.205     -0.077      0.938      -0.419       0.387
x233           0.7264      0.172      4.226      0.000       0.389       1.063
x234          -0.0139      0.009     -1.535      0.125      -0.032       0.004
==============================================================================
Omnibus:                      446.088   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            14790.331
Skew:                          -0.249   Prob(JB):                         0.00
Kurtosis:                      16.165   Cond. No.                          nan
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is -2.38e-14. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

```
/home/dominiq/anaconda3/lib/python3.10/site-packages/statsmodels/regression/linear_model.py:1965: Runti
meWarning: invalid value encountered in sqrt
  return np.sqrt(eigvals[0]/eigvals[-1])
```

In [ ]:

```python
from statsmodels.tools.tools import pinv_extended

model = sm.OLS(y_train, X_train)
results = model.fit_regularized(method="elastic_net", alpha=0, L1_wt=reg_ridge_.alpha_)

pinv_wexog,_ = pinv_extended(model.wexog)
normalized_cov_params = np.dot(pinv_wexog, np.transpose(pinv_wexog))
final = sm.regression.linear_model.OLSResults(model, results.params, normalized_cov_params)

rmse_train_ridge, rmse_test_ridge = get_rmse(results, X_train, X_test, y_train, y_test)

het_bp_ridge = het_breuschpagan(final.resid, X_train)
het_jb_ridge = jarque_bera(final.resid)

results_r_ridge = final.rsquared
print(final.summary())
```

[TRAIN] RMSE: 21077.46
[TEST] RMSE: 24441.16

OLS Regression Results

Dep. Variable:	SalePrice	R-squared:	0.903
Model:	OLS	Adj. R-squared:	0.890
Method:	Least Squares	F-statistic:	73.23
Date:	Tue, 17 Oct 2023	Prob (F-statistic):	0.00
Time:	10:39:04	Log-Likelihood:	1315.9
No. Observations:	2045	AIC:	-2170.
Df Residuals:	1814	BIC:	-870.9
Df Model:	230		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	11.8806	0.363	32.754	0.000	11.169	12.592
x1	-0.2367	0.158	-1.499	0.134	-0.546	0.073
x2	-0.0762	0.045	-1.688	0.092	-0.165	0.012
x3	0.0974	0.070	1.388	0.165	-0.040	0.235
x4	0.0392	0.140	0.280	0.780	-0.235	0.314
x5	0.0254	0.065	0.389	0.698	-0.103	0.154
x6	-0.1203	0.069	-1.751	0.080	-0.255	0.014
x7	-0.1157	0.099	-1.167	0.243	-0.310	0.079
x8	-0.1635	0.115	-1.424	0.155	-0.389	0.062
x9	-0.0081	0.078	-0.104	0.917	-0.161	0.145
x10	-0.0085	0.074	-0.116	0.908	-0.153	0.136
x11	-0.0428	0.075	-0.568	0.570	-0.191	0.105
x12	-0.0263	0.096	-0.274	0.784	-0.215	0.162
x13	0.1173	0.098	1.196	0.232	-0.075	0.310
x14	0.1629	0.083	1.964	0.050	0.000	0.326
x15	0.1822	0.038	4.790	0.000	0.108	0.257
x16	-0.2235	0.200	-1.117	0.264	-0.616	0.169
x17	0.0325	0.200	0.162	0.871	-0.360	0.425
x18	-0.1988	0.227	-0.875	0.382	-0.644	0.247
x19	-0.0175	0.201	-0.087	0.931	-0.412	0.377
x20	0.0193	0.197	0.098	0.922	-0.368	0.406
x21	-0.0061	0.197	-0.031	0.975	-0.392	0.379
x22	0.0272	0.026	1.065	0.287	-0.023	0.077
x23	-0.0432	0.032	-1.365	0.172	-0.105	0.019
x24	-0.0025	0.018	-0.138	0.891	-0.038	0.033
x25	0.0064	0.016	0.395	0.693	-0.026	0.038
x26	-0.0174	0.022	-0.791	0.429	-0.061	0.026
x27	-0.0052	0.043	-0.120	0.904	-0.091	0.080
x28	-0.0088	0.009	-1.009	0.313	-0.026	0.008
x29	0.0541	0.078	0.693	0.488	-0.099	0.207
x30	-0.0768	0.057	-1.356	0.175	-0.188	0.034
x31	-0.0652	0.046	-1.413	0.158	-0.156	0.025
x32	0.0039	0.047	0.081	0.935	-0.089	0.097
x33	0.0174	0.037	0.477	0.634	-0.054	0.089
x34	0.0284	0.042	0.669	0.503	-0.055	0.112
x35	-0.0999	0.040	-2.517	0.012	-0.178	-0.022
x36	0.0020	0.039	0.053	0.958	-0.074	0.078
x37	-0.0059	0.067	-0.088	0.930	-0.137	0.125
x38	-0.1771	0.051	-3.498	0.000	-0.276	-0.078
x39	-0.1621	0.058	-2.796	0.005	-0.276	-0.048
x40	-0.0054	0.040	-0.135	0.893	-0.083	0.073
x41	-0.0688	0.039	-1.761	0.078	-0.145	0.008
x42	0.0837	0.068	1.233	0.218	-0.049	0.217
x43	0.0053	0.040	0.131	0.896	-0.073	0.084
x44	0.1135	0.044	2.591	0.010	0.028	0.199
x45	0.0964	0.037	2.586	0.010	0.023	0.169
x46	-0.1693	0.047	-3.628	0.000	-0.261	-0.078
x47	-0.0627	0.050	-1.264	0.206	-0.160	0.035
x48	-0.0554	0.041	-1.361	0.174	-0.135	0.024
x49	0.0250	0.039	0.648	0.517	-0.051	0.101
x50	0.0953	0.044	2.152	0.032	0.008	0.182
x51	0.1218	0.042	2.873	0.004	0.039	0.205
x52	0.0355	0.040	0.877	0.380	-0.044	0.115
x53	0.0278	0.051	0.549	0.583	-0.072	0.127
x54	-0.0025	0.024	-0.105	0.917	-0.050	0.045
x55	0.0228	0.020	1.148	0.251	-0.016	0.062
x56	0.0671	0.045	1.499	0.134	-0.021	0.155
x57	0.0713	0.035	2.041	0.041	0.003	0.140
x58	-0.0101	0.038	-0.266	0.790	-0.084	0.064
x59	-0.0269	0.036	-0.739	0.460	-0.098	0.045
x60	-0.0365	0.084	-0.433	0.665	-0.202	0.129
x61	-0.0402	0.060	-0.673	0.501	-0.157	0.077
x62	-0.0510	0.093	-0.551	0.582	-0.232	0.130
x63	-0.0128	0.078	-0.164	0.870	-0.165	0.140

x64	0.0678	0.111	0.612	0.541	-0.149	0.285
x65	-0.0510	0.129	-0.395	0.693	-0.305	0.203
x66	-0.1782	0.237	-0.753	0.452	-0.642	0.286
x67	-0.1322	0.161	-0.823	0.411	-0.447	0.183
x68	0.0393	0.160	0.245	0.806	-0.275	0.354
x69	0.0308	0.127	0.243	0.808	-0.218	0.279
x70	0.0437	0.038	1.149	0.251	-0.031	0.118
x71	-0.0854	0.070	-1.224	0.221	-0.222	0.051
x72	-0.0463	0.065	-0.707	0.480	-0.175	0.082
x73	0.1065	0.093	1.139	0.255	-0.077	0.290
x74	-0.0276	0.043	-0.636	0.525	-0.113	0.057
x75	0.0669	0.087	0.767	0.443	-0.104	0.238
x76	0.1089	0.066	1.656	0.098	-0.020	0.238
x77	0.0534	0.040	1.324	0.186	-0.026	0.133
x78	-0.1579	0.059	-2.664	0.008	-0.274	-0.042
x79	-0.1073	0.079	-1.355	0.175	-0.263	0.048
x80	0.0107	0.079	0.135	0.892	-0.144	0.165
x81	0.0406	0.087	0.465	0.642	-0.130	0.212
x82	0.0135	0.079	0.170	0.865	-0.142	0.169
x83	-0.1012	0.099	-1.020	0.308	-0.296	0.093
x84	0.0924	0.162	0.569	0.569	-0.226	0.411
x85	0.0662	0.162	0.410	0.682	-0.251	0.383
x86	0.0410	0.162	0.252	0.801	-0.277	0.359
x87	-0.1403	0.146	-0.961	0.337	-0.427	0.146
x88	-0.0034	0.071	-0.048	0.962	-0.143	0.137
x89	0.0533	0.066	0.814	0.416	-0.075	0.182
x90	0.1241	0.087	1.419	0.156	-0.047	0.295
x91	0.0768	0.157	0.489	0.625	-0.231	0.385
x92	0.0218	0.093	0.235	0.815	-0.160	0.204
x93	0.1948	0.060	3.256	0.001	0.077	0.312
x94	-0.2052	0.234	-0.876	0.381	-0.665	0.254
x95	0.0370	0.115	0.321	0.748	-0.189	0.263
x96	0.1222	0.058	2.124	0.034	0.009	0.235
x97	0.0457	0.155	0.294	0.768	-0.259	0.350
x98	0.0368	0.072	0.512	0.609	-0.104	0.178
x99	0.1311	0.057	2.313	0.021	0.020	0.242
x100	0.6146	0.076	8.067	0.000	0.465	0.764
x101	-0.0163	0.129	-0.126	0.900	-0.269	0.237
x102	0.0876	0.067	1.311	0.190	-0.043	0.219
x103	0.1147	0.064	1.782	0.075	-0.012	0.241
x104	0.0955	0.056	1.693	0.091	-0.015	0.206
x105	0.1360	0.062	2.205	0.028	0.015	0.257
x106	0.0391	0.121	0.323	0.747	-0.198	0.276
x107	-0.0958	0.085	-1.126	0.260	-0.263	0.071
x108	-0.1020	0.064	-1.602	0.109	-0.227	0.023
x109	0.0647	0.116	0.560	0.575	-0.162	0.291
x110	-0.0776	0.058	-1.349	0.178	-0.190	0.035
x111	-0.1031	0.069	-1.489	0.137	-0.239	0.033
x112	0.0406	0.072	0.563	0.573	-0.101	0.182
x113	-0.1260	0.154	-0.818	0.413	-0.428	0.176
x114	-0.0762	0.056	-1.373	0.170	-0.185	0.033
x115	0	0.076	0	1.000	-0.149	0.149
x116	0.0867	0.101	0.858	0.391	-0.112	0.285
x117	-0.0050	0.066	-0.075	0.940	-0.135	0.125
x118	-0.0381	0.064	-0.593	0.553	-0.164	0.088
x119	-0.0422	0.056	-0.748	0.455	-0.153	0.068
x120	-0.0805	0.060	-1.342	0.180	-0.198	0.037
x121	-0.0286	0.034	-0.838	0.402	-0.096	0.038
x122	-0.4940	0.204	-2.418	0.016	-0.895	-0.093
x123	-0.0378	0.034	-1.103	0.270	-0.105	0.029
x124	-0.0097	0.036	-0.267	0.789	-0.080	0.061
x125	0.0219	0.015	1.448	0.148	-0.008	0.052
x126	0.0800	0.017	4.791	0.000	0.047	0.113
x127	-0.0629	0.043	-1.478	0.140	-0.146	0.021
x128	0.0552	0.068	0.808	0.419	-0.079	0.189
x129	0.0293	0.067	0.435	0.664	-0.103	0.161
x130	-0.0702	0.144	-0.487	0.626	-0.353	0.213
x131	-0.0072	0.148	-0.049	0.961	-0.297	0.282
x132	-0.2356	0.154	-1.525	0.127	-0.538	0.067
x133	-0.6048	0.183	-3.312	0.001	-0.963	-0.247
x134	0.1285	0.018	7.335	0.000	0.094	0.163
x135	-0.0032	0.029	-0.112	0.911	-0.059	0.053
x136	0.0509	0.068	0.743	0.457	-0.083	0.185
x137	-0.0168	0.015	-1.144	0.253	-0.046	0.012
x138	0.0327	0.141	0.232	0.816	-0.243	0.308
x139	-0.0680	0.038	-1.771	0.077	-0.143	0.007
x140	-0.0270	0.052	-0.521	0.603	-0.129	0.075
x141	-0.0147	0.042	-0.351	0.725	-0.096	0.067
x142	-0.2043	0.057	-3.555	0.000	-0.317	-0.092
x143	-0.0886	0.038	-2.314	0.021	-0.164	-0.013

x144	-0.2427	0.129	-1.879	0.060	-0.496	0.011
x145	-0.0618	0.138	-0.448	0.654	-0.333	0.209
x146	-0.0260	0.010	-2.699	0.007	-0.045	-0.007
x147	-0.0416	0.012	-3.596	0.000	-0.064	-0.019
x148	-0.0325	0.023	-1.425	0.154	-0.077	0.012
x149	-0.0140	0.019	-0.754	0.451	-0.051	0.022
x150	-0.0545	0.060	-0.914	0.361	-0.171	0.062
x151	-0.0167	0.017	-0.993	0.321	-0.050	0.016
x152	0.0136	0.130	0.105	0.916	-0.241	0.268
x153	0.0604	0.141	0.427	0.669	-0.217	0.338
x154	-0.0058	0.122	-0.048	0.962	-0.244	0.232
x155	-0.0632	0.198	-0.318	0.750	-0.452	0.326
x156	-0.0133	0.016	-0.856	0.392	-0.044	0.017
x157	0.0016	0.020	0.080	0.936	-0.037	0.040
x158	-0.0256	0.018	-1.437	0.151	-0.061	0.009
x159	-0.0063	0.018	-0.345	0.730	-0.042	0.030
x160	0.0036	0.013	0.277	0.782	-0.022	0.029
x161	-0.0148	0.013	-1.150	0.250	-0.040	0.010
x162	-0.0113	0.016	-0.717	0.473	-0.042	0.020
x163	-0.0013	0.013	-0.097	0.923	-0.028	0.025
x164	-0.0364	0.018	-2.036	0.042	-0.071	-0.001
x165	-0.0139	0.017	-0.822	0.411	-0.047	0.019
x166	-0.0138	0.017	-0.802	0.423	-0.047	0.020
x167	0.0692	0.049	1.416	0.157	-0.027	0.165
x168	0.1449	0.067	2.172	0.030	0.014	0.276
x169	0.0226	0.040	0.569	0.570	-0.055	0.100
x170	-0.0671	0.057	-1.180	0.238	-0.179	0.044
x171	-0.0381	0.076	-0.502	0.616	-0.187	0.111
x172	0.1267	0.070	1.818	0.069	-0.010	0.263
x173	0.1878	0.065	2.883	0.004	0.060	0.315
x174	-0.1035	0.141	-0.732	0.464	-0.381	0.174
x175	0.0116	0.020	0.586	0.558	-0.027	0.051
x176	0.2068	0.060	3.464	0.001	0.090	0.324
x177	0.0786	0.049	1.612	0.107	-0.017	0.174
x178	0.0607	0.028	2.134	0.033	0.005	0.116
x179	0.1128	0.014	7.957	0.000	0.085	0.141
x180	0.0504	0.067	0.758	0.449	-0.080	0.181
x181	0.0088	0.035	0.248	0.804	-0.060	0.078
x182	0.5379	0.114	4.737	0.000	0.315	0.761
x183	-0.1151	0.061	-1.876	0.061	-0.235	0.005
x184	0.0033	0.021	0.157	0.875	-0.038	0.044
x185	-0.0300	0.021	-1.441	0.150	-0.071	0.011
x186	0.0095	0.155	0.061	0.951	-0.294	0.313
x187	-0.0067	0.034	-0.198	0.843	-0.073	0.060
x188	0.5232	0.045	11.621	0.000	0.435	0.612
x189	0.0797	0.034	2.346	0.019	0.013	0.146
x190	-0.2775	0.055	-5.047	0.000	-0.385	-0.170
x191	0.1187	0.016	7.297	0.000	0.087	0.151
x192	0.1080	0.046	2.343	0.019	0.018	0.198
x193	0.0738	0.031	2.370	0.018	0.013	0.135
x194	-0.0413	0.039	-1.051	0.293	-0.118	0.036
x195	-0.0898	0.040	-2.218	0.027	-0.169	-0.010
x196	-0.0128	0.050	-0.258	0.796	-0.110	0.085
x197	0.0274	0.013	2.163	0.031	0.003	0.052
x198	0.0325	0.015	2.127	0.034	0.003	0.062
x199	0.4593	0.029	15.901	0.000	0.403	0.516
x200	-0.0659	0.035	-1.902	0.057	-0.134	0.002
x201	0.3311	0.045	7.399	0.000	0.243	0.419
x202	0.2699	0.025	10.748	0.000	0.221	0.319
x203	-0.0422	0.036	-1.175	0.240	-0.113	0.028
x204	-0.0328	0.019	-1.758	0.079	-0.069	0.004
x205	0.5958	0.047	12.707	0.000	0.504	0.688
x206	0.3147	0.045	7.014	0.000	0.227	0.403
x207	0.1411	0.079	1.785	0.074	-0.014	0.296
x208	-0.0462	0.042	-1.094	0.274	-0.129	0.037
x209	-0.0380	0.028	-1.359	0.174	-0.093	0.017
x210	-0.0298	0.028	-1.046	0.296	-0.086	0.026
x211	-0.1690	0.043	-3.970	0.000	-0.252	-0.086
x212	-0.0241	0.021	-1.163	0.245	-0.065	0.017
x213	-0.4432	0.053	-8.377	0.000	-0.547	-0.339
x214	-0.3769	0.072	-5.220	0.000	-0.518	-0.235
x215	0.1523	0.031	4.871	0.000	0.091	0.214
x216	0.3349	0.050	6.706	0.000	0.237	0.433
x217	-0.0970	0.039	-2.479	0.013	-0.174	-0.020
x218	0.1062	0.026	4.021	0.000	0.054	0.158
x219	-0.0724	0.026	-2.824	0.005	-0.123	-0.022
x220	0.0423	0.033	1.276	0.202	-0.023	0.107
x221	0.0765	0.044	1.743	0.082	-0.010	0.163
x222	0.0368	0.056	0.658	0.511	-0.073	0.147
x223	-0.2367	0.087	-2.728	0.006	-0.407	-0.067

```
x224          -0.0048      0.093     -0.052      0.959      -0.187       0.178
x225           0.0764      0.016      4.714      0.000       0.045       0.108
x226           0.0874      0.040      2.162      0.031       0.008       0.167
x227           0.0376      0.041      0.921      0.357      -0.042       0.118
x228           0.1487      0.057      2.614      0.009       0.037       0.260
x229           0.0857      0.062      1.386      0.166      -0.036       0.207
x230           0.1274      0.034      3.800      0.000       0.062       0.193
x231          -0.0472      0.205     -0.230      0.818      -0.449       0.354
x232          -0.0587      0.227     -0.259      0.795      -0.503       0.386
x233           0.1071      0.190      0.565      0.572      -0.265       0.479
x234          -0.0197      0.010     -1.976      0.048      -0.039      -0.000
==============================================================================
Omnibus:                      716.075   Durbin-Watson:                   1.956
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            11965.007
Skew:                          -1.197   Prob(JB):                         0.00
Kurtosis:                      14.606   Cond. No.                          nan
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is -2.38e-14. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

```
/home/dominiq/anaconda3/lib/python3.10/site-packages/statsmodels/regression/linear_model.py:1965: Runti
meWarning: invalid value encountered in sqrt
  return np.sqrt(eigvals[0]/eigvals[-1])
```

In [ ]:
```
df_compare = pd.DataFrame({
    'MRS_TRAIN': [rmse_train_ols, rmse_train_ridge, rmse_train_lasso],
    'MRS_TEST':[rmse_test_ols, rmse_test_ridge, rmse_test_lasso],
    'R-squared': [results_r_ols, results_r_ridge, results_r_lasso],
    'Jarque-Bera (p_value)': [het_jb_ols[1], het_jb_ridge[1], het_jb_lasso[1]],
    'Breusch-Pagan (p_value)': [het_bp_ols[1], het_bp_ridge[1], het_bp_lasso[1]],
})

df_compare.index =  ['OLS', 'OLS_Ridge', 'OLS_Lasso']

with pd.option_context('float_format', '{:.3f}'.format, 'display.expand_frame_repr', False):
    display(df_compare)
```

	MRS_TRAIN	MRS_TEST	R-squared	Jarque-Bera (p_value)	Breusch-Pagan (p_value)
OLS	16745.597	21323.383	0.940	0.000	0.000
OLS_Ridge	21077.464	24441.163	0.903	0.000	0.000
OLS_Lasso	18507.507	21326.870	0.920	0.000	0.000