Video Game Sales Analysis

DC127235 Yang Zheyu
DC127018 Shi Jiayu
DC126940 Yang Dishen
DC127244 Chen Risheng
DC127217 Xu Wenxin

TABLE OF CONTENTS

01

Introduction of Project

Introduction

Data Source: Kaggle “Video Game Sales with Ratings” Data set

Including:

The name of the game, the platform, the year of release, the type of game, the publisher, the sales volume, and the rating.

Why?

Trying to make a model to predict sales.

Ridge & Lasso & Elastic Net Regression & Random Forest

02

Data Preprocessing & Visualization

Data Sets Overview

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count	Developer	Rating
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	3.77	8.45	82.53	76.0	51.0	8	322.0	Nintendo	E
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24	NaN	NaN	NaN	NaN	NaN	NaN
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	3.79	3.29	35.52	82.0	73.0	8.3	709.0	Nintendo	E
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	3.28	2.95	32.77	80.0	73.0	8	192.0	Nintendo	E
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37	NaN	NaN	NaN	NaN	NaN	NaN

Data Information

```
# information about vgsales
vgsales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16719 entries, 0 to 16718
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Name             16717 non-null  object
 1   Platform         16719 non-null  object
 2   Year_of_Release  16450 non-null  float64
 3   Genre            16717 non-null  object
 4   Publisher        16665 non-null  object
 5   NA_Sales         16719 non-null  float64
 6   EU_Sales         16719 non-null  float64
 7   JP_Sales         16719 non-null  float64
 8   Other_Sales      16719 non-null  float64
 9   Global_Sales     16719 non-null  float64
 10  Critic_Score     8137 non-null   float64
 11  Critic_Count     8137 non-null   float64
 12  User_Score       10015 non-null  object
 13  User_Count       7590 non-null   float64
 14  Developer        10096 non-null  object
 15  Rating           9950 non-null   object
dtypes: float64(9), object(7)
memory usage: 2.0+ MB
```

Missing Value

```
# missing value of each column
vgsales.isnull().sum()
```

```
Name                 2
Platform             0
Year_of_Release    269
Genre                2
Publisher           54
NA_Sales             0
EU_Sales             0
JP_Sales             0
Other_Sales          0
Global_Sales         0
Critic_Score      8582
Critic_Count      8582
User_Score        6704
User_Count        9129
Developer         6623
Rating            6769
dtype: int64
```

Descriptive Statistical Analysis

```
# descriptive statistical analysis
vgsales.describe()
```

	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critical_Score	Critical_Count	User_Score	User_Count
count	6825	6825	6825	6825	6825	6825	6825	6825	6825	6825
mean	2007.436777	0.394484	0.236089	0.064158	0.082677	0.77759	70.272088	28.931136	7.185626	174.722344
std	4.211248	0.967385	0.68733	0.28757	0.269871	1.963443	13.868572	19.224165	1.439942	587.428538
min	1985	0	0	0	0	0.01	13	3	0.5	4
25%	2004	0.06	0.02	0	0.01	0.11	62	14	6.5	11
50%	2007	0.15	0.06	0	0.02	0.29	72	25	7.5	27
75%	2011	0.39	0.21	0.01	0.07	0.75	80	39	8.2	89
max	2016	41.36	28.96	6.5	10.57	82.53	98	113	9.6	10665

Data Preprocessing

Missing Value Handling

```
# drop the few null values
vgsales.dropna(inplace=True)x

# display the new number of row
vgsales.shape[0]
```

```
6825
```

Data Type Adjustments

- 'Year_of_Release' is converted to integer type.
- 'User_Score' is converted from object to numeric type.

```
# convert Year_of_Release from float to int
vgsales['Year_of_Release']=vgsales['Year_of_Release'].astype(int)

# convert User_Score from object to float
vgsales['User_Score']=vgsales['User_Score'].astype(float)
```

Data Preprocessing(cont.)

One-Hot Encoding

- Categorical variable 'Genre', 'Platform', 'Publisher' and 'Rating' are one-hot encoded

```
# one-hot encoding with get_dummies allows to map each feature with the coefficient of the Ridge regression
vgRidge = pd.concat([pd.get_dummies(vgRidge[['Genre','Platform','Publisher','Rating']]),
                     vgRidge[['Year_of_Release','Global_Sales','Critic_Score','Critic_Count','User_Score','User_Count']]],
                    axis = 1)
```

Data Standardization

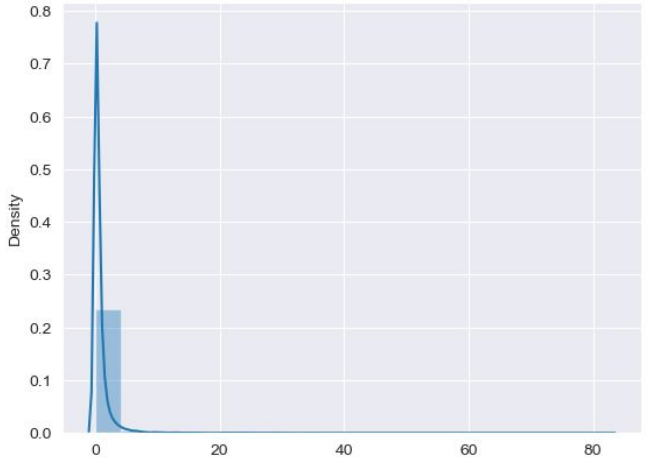
- Define and standardize numerical variables

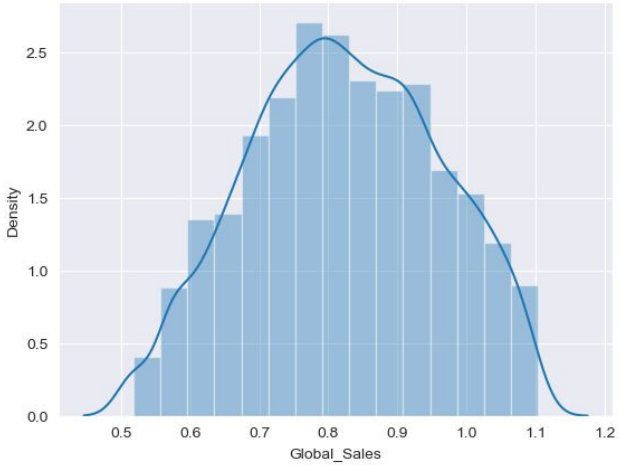
```
# define the columns to be standardized
numeric_columns = vgsales.columns[11:14]

# standardization of numerical variables
vgsales[numeric_columns] = scaler.fit_transform(vgsales[numeric_columns])
```

Data Preprocessing(cont.)

Extreme Values Handling

- The distribution is right-skewed.
- Take only the left tail of the distribution, which contains most of the entries.
- Normal distribution obtained

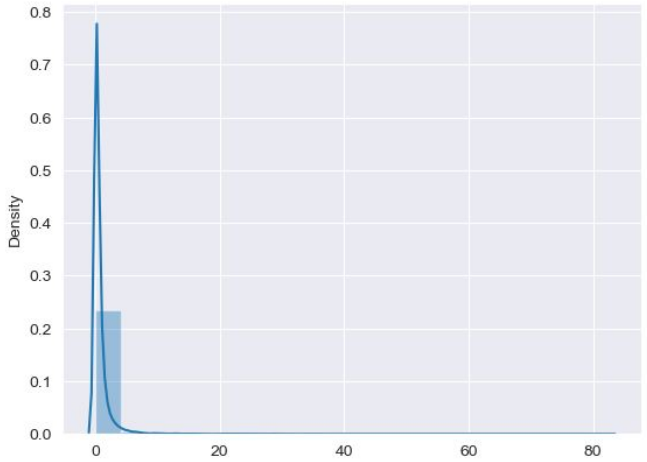


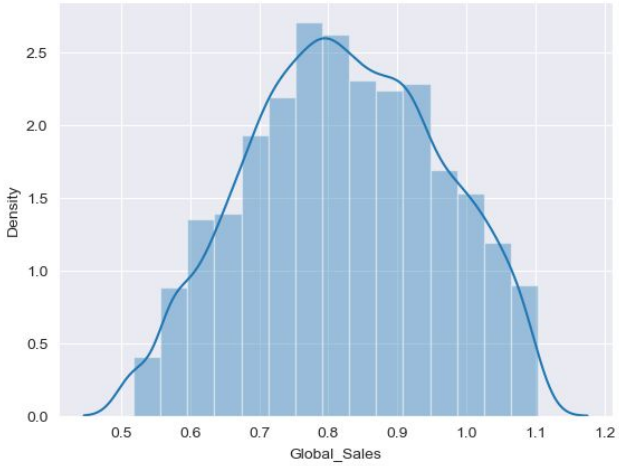


Data Preprocessing(cont.)

Extreme Values Handling

```
# take only left tail of the distribution
restrictionSales = (vgRidge['Global_Sales'] < 2)
# plot histogram of Sales with the power 1/7 to obtain normal distribution
sns.distplot((vgRidge['Global_Sales'][restrictionSales])**(1/7),bins = 15,kde = True)
plt.show()
```





Data visualization

- **Number of games of various types**

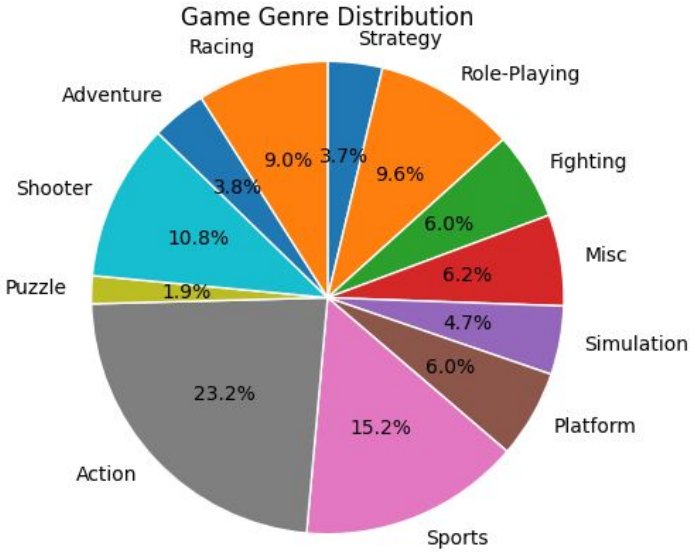
```python
import pandas as pd

# Read CSV file
data = pd.read_csv('new_data_without_outliers.csv')

# Count the number of games of various types
genre_counts = data['Genre'].value_counts()

# Print results
print(genre_counts)
```

```
Action          1411
Sports           923
Shooter          656
Role-Playing     584
Racing           547
Misc             380
Platform         366
Fighting         365
Simulation       286
Adventure        229
Strategy         228
Puzzle           116
Name: Genre, dtype: int64
```

Data visualization

- **Sales proportion of various games**

```
# Count the number of games of various types
genre_counts = data['Genre'].value_counts()

# Shuffle game type data randomly
genre_shuffled = genre_counts.sample(frac=1)

# Create canvases and subplots
fig, ax = plt.subplots()
```



Data visualization

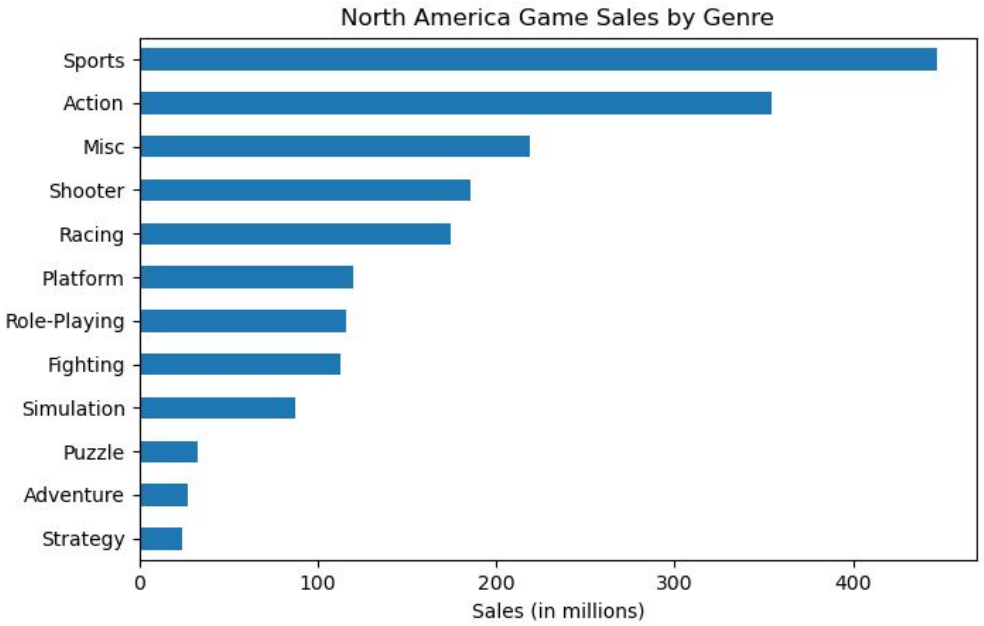
- **Game sales in different markets**

Genre	NA_Sales	EU_Sales	JP_Sales	Other_Sales
Action	354.36	210.72	49.82	72.95
Adventure	27.22	14.41	8.01	4.48
Fighting	113.18	50.82	21.62	21.70
Misc	218.83	119.12	32.75	39.99
Platform	120.10	61.17	17.00	19.15
Puzzle	32.97	23.63	14.79	6.23
Racing	174.30	116.46	14.26	45.46
Role-Playing	116.35	45.96	86.79	17.49
Shooter	185.67	96.72	7.05	33.71
Simulation	87.57	55.30	21.77	15.66
Sports	446.93	214.91	33.77	86.76
Strategy	24.39	12.02	4.01	3.87

```
Process finished with exit code 0
```

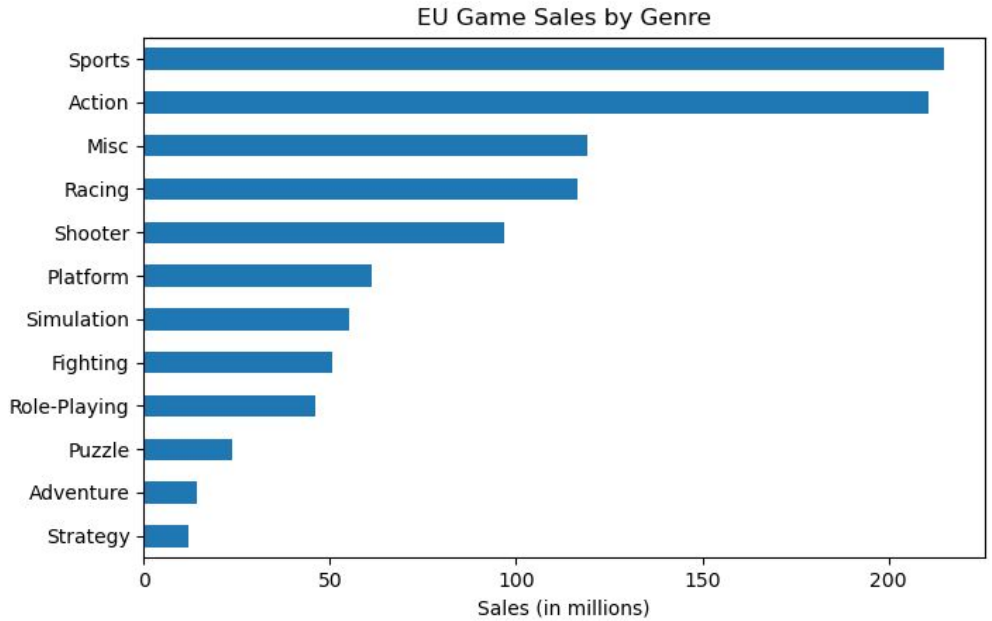
Data visualization

- **North American market game sales**



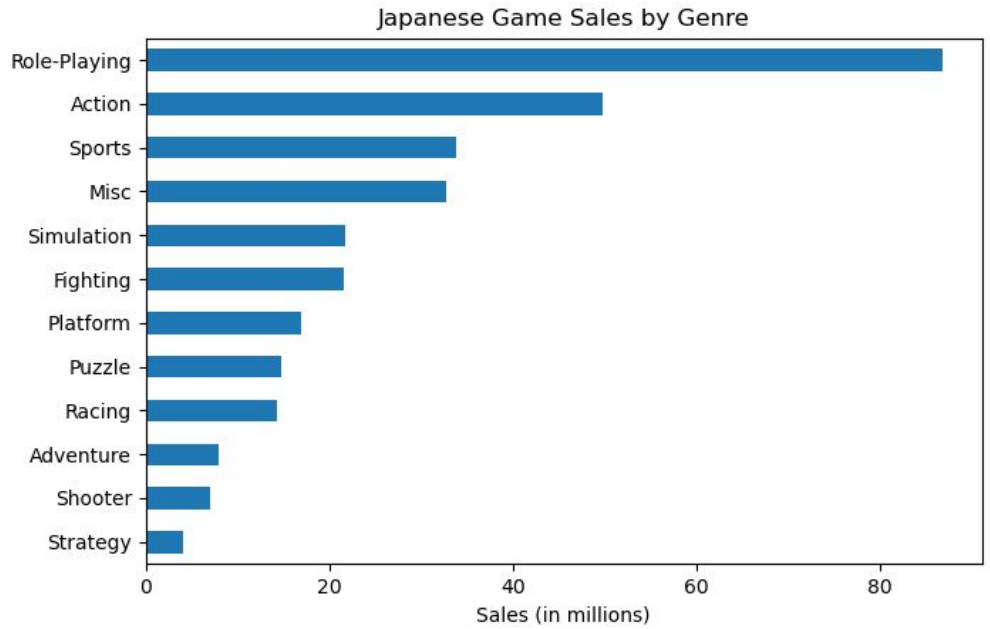
Data visualization

- **European market game sales**



Data visualization

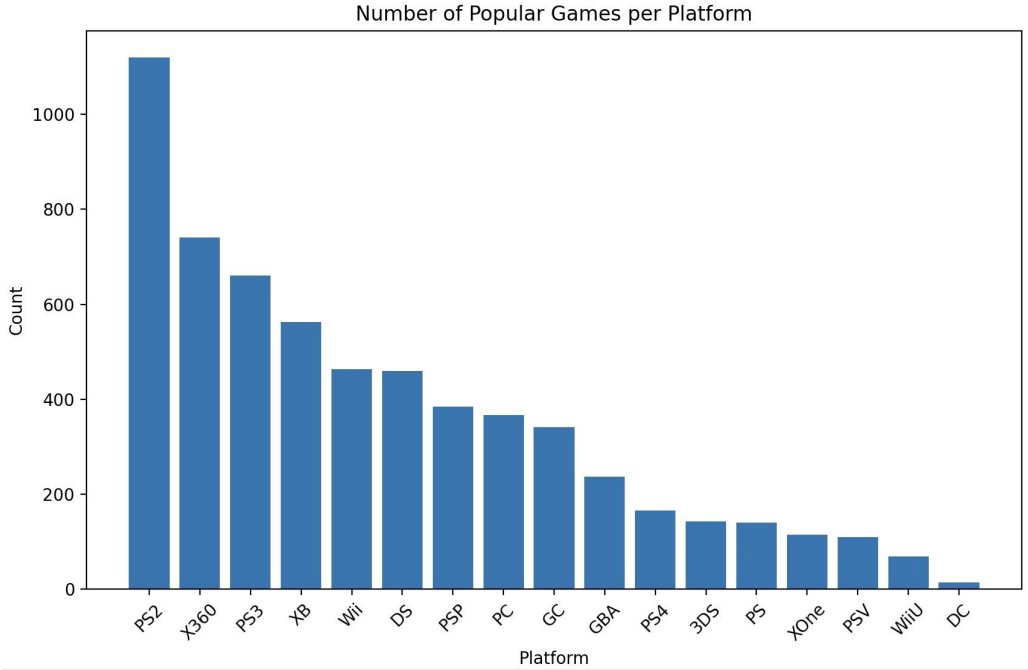
- **Japanese market game sales**



Data visualization

- **Number of popular games on each platform**

Platform	Count
PS2	1120
X360	741
PS3	660
XB	562
Wii	463
DS	460
PSP	384
PC	367
GC	341
GBA	237
PS4	166
3DS	143
PS	140
XOne	114
PSV	110
WiiU	69
DC	14



Data visualization

- **Sales of Global Gaming Companies**

Publisher	Global Sales (in millions)
Electronic Arts	676.09
Nintendo	547.72
Activision	294.70
Sony Computer Entertainment	232.10
Ubisoft	219.24
THQ	151.79
Take-Two Interactive	140.86
Sega	135.81
Konami Digital Entertainment	97.77
Namco Bandai Games	96.88

Data visualization

- **Sales of different distribution companies in each market segment**

North America

Publisher	NA_Sales
Electronic Arts	394.99
Nintendo	234.56
Activision	185.20
Ubisoft	122.22
Sony Computer Entertainment	108.60
...	...
Pinnacle	0.00
Revolution Software	0.00
Russel	0.00
Strategy First	0.00
bitComposer Games	0.00

Europe

Publisher	EU_Sales
Electronic Arts	195.53
Nintendo	156.03
Activision	79.08
Ubisoft	70.92
Sony Computer Entertainment	65.60
...	...
Destineer	0.00
Irem Software Engineering	0.00
DSI Games	0.00
DHM Interactive	0.00
Nobilis	0.00

Japan

Publisher	JP_Sales
Nintendo	117.27
Square Enix	25.53
Sony Computer Entertainment	25.23
Capcom	20.33
Namco Bandai Games	20.15
...	...
Introversion Software	0.00
Jaleco	0.00
Jester Interactive	0.00
Just Flight	0.00
bitComposer Games	0.00

03

Ridge & Lasso & Elastic Net Regression

Ridge Regression

- Ridge regression is a model tuning method that is used to analyze any data that suffers from multicollinearity.
- This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.
- The objective is to minimize the sum of squared errors between the predicted values and the actual values,

$$\min_{\theta} \|\mathbf{y} - \mathbf{x}\theta\|^2 + \lambda\|\theta\|^2$$

Ridge Regression

- Denote the data $((y_1, x_1), \ldots, (y_n, x_n))$, where $y_i \in \mathbb{R}$ and $x_i \in \mathbb{R}^d$ for $i = 1, \ldots, n$. Write $y = [y_1, \ldots, y_n]^T \in \mathbb{R}^n$ and $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d \times n}$.

- We consider the linear regression:

$$y_i = x_i' \theta + \epsilon_i, \quad i = 1, 2, \ldots, n, \tag{1}$$

with the restriction: for some $K > 0$,

$$\|\theta\|_2 = \sqrt{\theta_1^2 + \ldots + \theta_d^2} \leq K. \tag{2}$$

- To estimate the parameter θ, we solve the following optimization problem with constraint:

$$\min_{\theta} \|y - X'\theta\|^2, \tag{3}$$

with constraint

$$\|\theta\|_2 = \sqrt{\theta_1^2 + \ldots + \theta_d^2} \leq K. \tag{4}$$

Ridge Regression

- By Lagrangian multiplier method, to estimate the parameter θ, we solve the following optimization problem:

$$\min_{\theta} \|y - X\theta\|^2 + \lambda\|\theta\|^2 \tag{1}$$

  where $\lambda > 0$ is some tuning parameter to be chosen.

- Then we derive from (1):

$$\hat{\theta} = (\lambda I_d + X'X)^{-1}X'y, \tag{2}$$

  where I_d is the $d \times d$ identity matrix.

- According to Condition 3 of KKT:

$$\lambda \geq 0. \tag{3}$$

- According to Condition 4 of KKT:

$$\lambda(\|\hat{\theta}\|^2 - K) = 0. \tag{4}$$

Ridge Regression

- Condition 4 of KKT:

$$\lambda(\|\hat{\theta}\|^2 - K) = 0. \tag{1}$$

- If $X'X$ is invertible, the classical least square regression is

$$\hat{\theta}_{LS} = (X'X)^{-1}X'y. \tag{2}$$

  - $\|\hat{\theta}_{LS}\| \geq \|\hat{\theta}\|$.
  - If $K \geq \|\hat{\theta}_{LS}\|^2$, we have $\lambda = 0$. This means that the constraint $\|\theta\|^2 \leq K$ has no effect.

- In the case $\|\hat{\theta}\|^2 - K = 0$,

  - solve $\|\hat{\theta}\|^2 - K = 0$ to determine λ^* and then $\hat{\theta}^*$,
  - the smaller the K is, the larger the $\lambda > 0$ has to be chosen.

Lasso Regression

- Denote the data $(y_1, x_1), ..., (y_n, x_n)$,where $y_i \in R$ and $x_i \in R^d$ for $i = 1, ..., n$. Write $\boldsymbol{y} = [y_1, ..., y_n]' \in R^d$ and $\mathbf{X} = [x_1, ..., x_n] \in R^{p \times n}$.
- We consider the linear regression:

$$y_i = x_i'\theta + \varepsilon_i, \quad i = 1, 2, ..., n,$$

With the restriction: for some $K > 0$,

$$\|\theta\|_1 = |\theta_1| + ... + |\theta_d| \leqslant K.$$

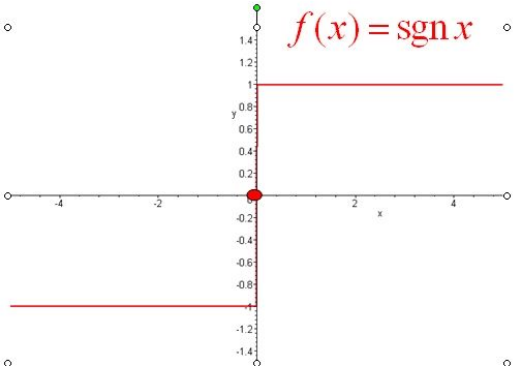
- To estimate the parameter θ, we solve the following optimization problem with constraint:

$$\min_{\theta} \|\boldsymbol{y} - \mathbf{X}'\theta\|^2,$$

with constraint

$$\|\theta\|_1 \leqslant K.$$

Lasso Regression



- The Lagrangian multiplier is

$$L(\theta, \lambda) = \|\boldsymbol{y} - \mathbf{X}'\theta\|^2 + \lambda\left(\|\theta\|_1 - K\right),$$

where $\lambda \geqslant 0$ is some tuning parameter to be chosen.

- We have

$$2\mathbf{X}\mathbf{X}'\theta = 2\mathbf{X}\boldsymbol{y} - \lambda \operatorname{sgn}(\theta).$$

where $\operatorname{sgn}(\theta) = [\operatorname{sgn}(\theta_1), \ldots, \operatorname{sgn}(\theta_p)]'$, and $\operatorname{sgn}(x) = 1$ if $x > 0$, $\operatorname{sgn}(x) = -1$ if $x < 0$, $\operatorname{sgn}(x) = 0$ if $x = 0$.

- If $\mathbf{X}'\mathbf{X}$ is invertible, So we have

$$\hat{\theta} = (\mathbf{X}\mathbf{X}')^{-1}\mathbf{X}\boldsymbol{y} - \frac{1}{2}\lambda(\mathbf{X}\mathbf{X}')^{-1}\operatorname{sgn}\left(\hat{\theta}\right) = \hat{\theta}^{ls} - \frac{1}{2}\lambda(\mathbf{X}\mathbf{X}')^{-1}\operatorname{sgn}\left(\hat{\theta}\right),$$

where $\hat{\theta}^{ls} = (\mathbf{X}\mathbf{X}')^{-1}\mathbf{X}\mathbf{y}$ is the least square estimator.

Lasso Regression

- To simplify our analysis, we consider a special case $\mathbf{X}\mathbf{X}' = Id$ and have

$$\hat{\theta} = \hat{\theta}^{ls} - \frac{1}{2}\lambda \operatorname{sgn}\left(\hat{\theta}\right).$$

More precisely,

$$\hat{\theta}_j = \hat{\theta}_j{}^{ls} - \frac{1}{2}\lambda \operatorname{sgn}\left(\hat{\theta}_j\right), \quad j = 1, \ldots, p. \tag{1}$$

- If $\hat{\theta}_j^{ls} > 0$, as $\lambda > 0$ is small, we have $\hat{\theta}_j > 0$. Let λ increase to $\lambda = 2\hat{\theta}_j^{ls}$, So we have

$$\hat{\theta}_j = 0.$$

Let λ continue increasing, we still have

$$\hat{\theta}_j = 0.$$

Indeed, if $\hat{\theta}_j < 0$, by (1) we have $\hat{\theta}_j > 0$. Contradiction! If $\hat{\theta}_j > 0$, by (1) we have $\hat{\theta}_j < 0$. Contradiction!

Elastic Net

- The elastic network model is as follows:

$$L_{enet}(\hat{\beta}) = \frac{\sum_{i=1}^{n}(y_i - x_i'\hat{\beta})^2}{2n} + \lambda\left(\frac{1-\alpha}{2}\sum_{j=1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j=1}^{n}|\hat{\beta}_j|\right)$$

The λ parameter is the regularization penalty, and α and β are two terms parameter.

- α = 0 corresponds to the ridge regression model, α = 1 corresponds to Lasso regression model
- By setting a specific α to optimize the model, on the one hand, the stability of ridge

regression can be inherited, and on the other hand, group effects can be performed in the case of highly correlated variables.

04

Model & Code

Model

Model

Plain Text

```
results = search.fit(X, y)
feature_coefs = model.coef_
top_10_indices = np.argsort(feature_coefs)[-10:][::-1]
top_10_coefs = feature_coefs[top_10_indices]
top_10_features = vgRidge.columns[top_10_indices]
model = Ridge(alpha = results.best_params_['alpha'])
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

Perform search

Plain Text

```
results_summary[sales_col]=
{
'r2': results.best_score_,
'Config': results.best_params_,
'MSE': mean_squared_error(y_test, model.predict(X_test)),
'Top 10 Coefs': top_10_coefs,
'Top 10 Features': top_10_features
}
```

Update result dictionary

Plain Text

```
for sales_col, metrics in results_summary.items():
print(f"Results for {sales_col}:")
print(f"r2: {metrics['r2']:.3f}")
print(f"Config: {metrics['Config']}")
print(f"MSE: {metrics['MSE']}")
print("Top 10 Coefs:")
for feature, coef in zip(metrics['Top 10 Features'], metrics['Top 10 Coefs']):
print(f"{feature}: {coef}")
print("----------------------------------------------------------")
```

Predict result output

Plain Text

```
model = Ridge(alpha=1.0)
```

Definition model

Plain Text

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1234)
```

Partition data set

Plain Text

```
model.fit(X_train, y_train)
```

Fitting model

Plain Text

```
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
grid = dict()
grid['alpha'] = np.logspace(-1, 1, num=50)
search = GridSearchCV(model, grid, scoring='r2', cv=cv, n_jobs=-1)
```

Define model evaluation methods

Cross-Validation

LOOCV:（Leave-one-out cross-validation）

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{MSE}_i. \qquad \mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2$$

K-fold Cross Validation

Our model

- Divide all data sets into 5 parts
- Take one copy of the test set and four copies of the training set each time
- Training the model without repeating
- Calculate the MSEi of the model on the test set
- Average five times to get the final MSEi

Tuning the alpha

```
# Define model evaluation method
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
grid = dict()
grid['alpha'] = np.logspace(-1, 1, num=50)
search = GridSearchCV(model, grid, scoring='r2', cv=cv, n_jobs=-1)

# Perform search
results = search.fit(X, y)
```

Alpha after tuning

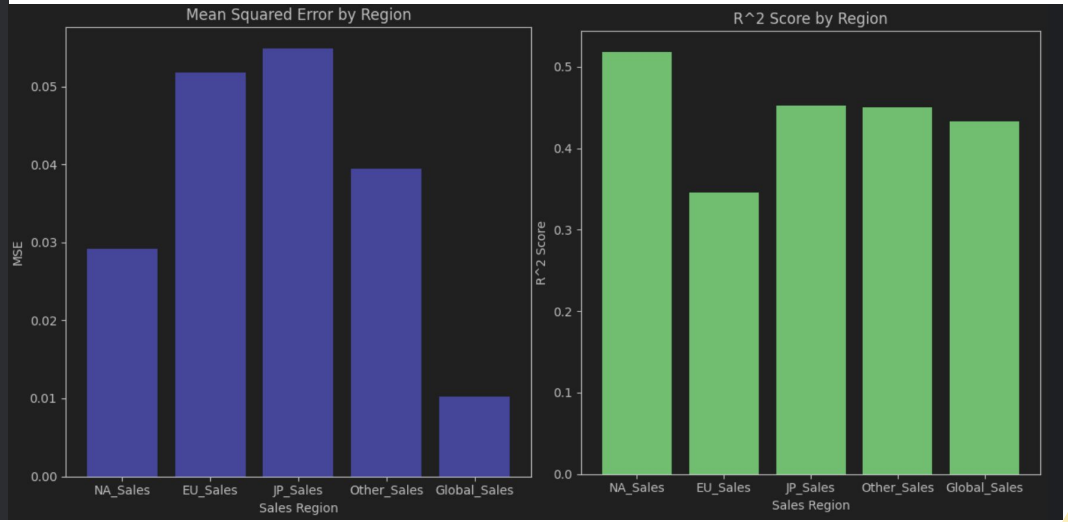
```
results_summary[sales_col] = {

    'r2': results.best_score_,
    'Config': results.best_params_,
    'MSE': mean_squared_error(y_test, model.predict(X_test)),
    'Top 10 Coefs': top_10_coefs,
    'Top 10 Features': top_10_features
}
```

```
# output result
for sales_col, metrics in results_summary.items():
    print(f"Results for {sales_col}:")
    print(f"r2: {metrics['r2']:.3f}")
    print(f"Config: {metrics['Config']}")
    print(f"MSE: {metrics['MSE']}")
    print("Top 10 Coefs:")
    for feature, coef in zip(metrics['Top 10 Features'], metrics['Top 10 Coefs']):
        print(f"{feature}: {coef}")
    print("----------------------------------------------------")
```

	R^2	MSE	Alpha after tunning
NA_Sales	0.518	0.02917018	4.29
EU_Sales	0.252	0.05938991	1.389495494
JP_Sales	0.391	0.06086096	2.222996483
Other_Sales	0.364	0.046861063	2.947051703
Global_Sales	0.29	0.013041954	2.682695795

Visualize the R^2 and MSE

```python
import matplotlib.pyplot as plt

mse_values = []
r2_values = []
sales_regions = []

for sales_col, metrics in results_summary.items():
    sales_regions.append(sales_col)
    mse_values.append(metrics['MSE'])
    r2_values.append(metrics['r2'])

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.bar(sales_regions, mse_values, color='blue')
plt.title('Mean Squared Error by Region')
plt.xlabel('Sales Region')
plt.ylabel('MSE')

plt.subplot(1, 2, 2)
plt.bar(sales_regions, r2_values, color='green')
plt.title('R^2 Score by Region')
plt.xlabel('Sales Region')
plt.ylabel('R^2 Score')

plt.tight_layout()
plt.show()
```



10 Largest coefficient

```
# Get the index of the largest 10 coefs
top_10_indices = np.argsort(feature_coefs)[-10:][::-1]

# Get the largest 10 coefs and corresponding feature names
top_10_coefs = feature_coefs[top_10_indices]
top_10_features = vgRidge.columns[top_10_indices]
```

NA	coefficient	Japan	coefficient
Publisher_Visco	0.229758387	Publisher_Nintendo	0.446345404
Publisher_1C Company	0.204971822	Publisher_From Software	0.427927797
Publisher_Trion Worlds	0.194105873	Publisher_Ubisoft Annecy	0.419372568
Publisher_Sony Computer Entertainment America	0.193951805	Publisher_Enix Corporation	0.413087685
Publisher_Square Enix	0.188671535	Publisher_SquareSoft	0.350990548
Publisher_Warner Bros. Interactive Entertainment	0.179299924	Publisher_Marvelous Entertainment	0.326428462
Publisher_LucasArts	0.177043433	Publisher_Gamebridge	0.32191309
Publisher_Activision Blizzard	0.166155012	Platform_DC	0.3003811
Publisher_SquareSoft	0.165063268	Publisher_Pacific Century Cyber Works	0.290936582
		Publisher_Marvelous Interactive	0.283676896

coefficient of numerical data

NA	
Year_of_Release	0.001048367
Critic_Score	0.002479765
Critic_Count	0.036936232
User_Score	-0.009051781
User_Count	0.029648732

Japan	
Year_of_Release	0.001048367
Critic_Score	0.002479765
Critic_Count	0.036936232
User_Score	-0.009051781
User_Count	0.029648732

PCA Test

PCA (Principal Component Analysis) as a method for reducing the dimensionality of a data set in which there is a large number of interrelated variables while retaining as much as possible the variation in the original set of variables

$$\mathbf{X}_{1\times p}^{\top} = (X_1, X_2, \ldots, X_p) \quad \text{to} \quad \mathbf{Y}_{1\times q}^{\top} = (Y_1, Y_2, \ldots, Y_q)$$

$$\begin{aligned} Y_1 &= \mathbf{a}_1^{\top}\mathbf{X} = a_{11}X_1 + a_{12}X_2 + \cdots + a_{1p}X_p \\ Y_2 &= \mathbf{a}_2^{\top}\mathbf{X} = a_{21}X_1 + a_{22}X_2 + \cdots + a_{2p}X_p \\ &\vdots \\ Y_p &= \mathbf{a}_p^{\top}\mathbf{X} = a_{p1}X_1 + a_{p2}X_2 + \cdots + a_{pp}X_p \end{aligned}$$

$$\operatorname{var}(Y_i) = \mathbf{a}_i^{\top}\boldsymbol{\Sigma}_X\mathbf{a}_i$$

PCA Test

- find $\mathbf{a}_1$ that maximizes $\mathbf{a}_1^\top \mathbf{\Sigma}_X \mathbf{a}_1$ subject to $\mathbf{a}_1^\top \mathbf{a}_1 = 1$; or equivalently, find $\mathbf{a}_1$ that maximizes variance $\max_{\mathbf{a}_1} \left(\frac{\mathbf{a}_1^\top \mathbf{\Sigma}_X \mathbf{a}_1}{\mathbf{a}_1^\top \mathbf{a}_1} \right) = \text{var}(Y_1)$
- find $\mathbf{a}_2$ that maximizes $\mathbf{a}_2^\top \mathbf{\Sigma}_X \mathbf{a}_2$ subject to $\mathbf{a}_2^\top \mathbf{\Sigma}_X \mathbf{a}_1 = 0$ and $\mathbf{a}_2^\top \mathbf{a}_2 = 1$; or equivalently $\max_{\mathbf{a}_2 \perp \mathbf{a}_1} \left(\frac{\mathbf{a}_2^\top \mathbf{\Sigma}_X \mathbf{a}_2}{\mathbf{a}_2^\top \mathbf{a}_2} \right) = \text{var}(Y_2)$
- $\cdots$
- find $\mathbf{a}_i$ that maximizes $\mathbf{a}_i^\top \mathbf{\Sigma}_X \mathbf{a}_i$ subject to $\mathbf{a}_i^\top \mathbf{\Sigma}_X \mathbf{a}_k = 0$ for $k < i$ and $\mathbf{a}_i^\top \mathbf{a}_i = 1$;

Output:
Top 3 features for Principal Component 1: ['Publisher_Activision' 'Publisher_Ubisoft' 'Publisher_Electronic Arts']
Top 3 features for Principal Component 2: ['Publisher_Electronic Arts' 'Publisher_Activision' 'Publisher_Ubisoft']
Top 3 features for Principal Component 3: ['Publisher_THQ' 'Publisher_Ubisoft' 'Publisher_Activision']

Lasso regression

```
# Define model evaluation method
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
grid = dict()
grid['alpha'] = np.logspace(-1, 1, num=50)
search = GridSearchCV(model, grid, scoring='r2', cv=cv, n_jobs=-1)

# Perform search
results = search.fit(X, y)
```

→ Not converge!

```
# Define model evaluation method
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
grid = dict()
grid['alpha'] = np.logspace(-0.5, stop: 0.0005, num: 50)
search = GridSearchCV(model, grid, scoring='r2', cv=cv, n_jobs=-1)

# Perform search
results = search.fit(X, y)
```

→ Tuning the alpha in smaller logspace

Lasso regression

```
grid['alpha'] = np.logspace(-0.5, stop: 0.0005, num: 50)
```

	R^2	MSE	Alpha after tunning
NA_Sales	0.001	0.06603032	0.31622776
EU_Sales	0.076	0.0721617	0.31622776
JP_Sales	0.045	0.09469392	0.31622776
Other_Sales	0.048	0.07277833	0.31622776
Global_Sales	0.049	0.01857636	0.31622776

Elastic Net

```
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
grid = dict()
grid['l1_ratio']=np.linspace( start: 0.00000000005, stop: 0.00000000006, num: 9)
search = GridSearchCV(model, grid, scoring='r2', cv=cv, n_jobs=-1)
```

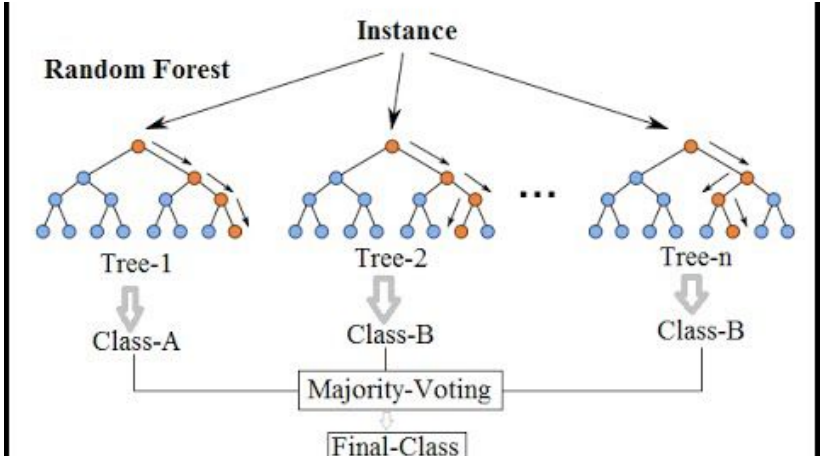
	R^2	MSE	Ratio after tunning
NA_Sales	0.134	0.05694065	5.00E-11
EU_Sales	0.157	0.06561083	5.00E-11
JP_Sales	0.158	0.0823455	5.00E-11
Other_Sales	0.17	0.06344067	5.00E-11
Global_Sales	0.17	0.01599804	5.00E-11

05

Classification of Sales category using Random Forest

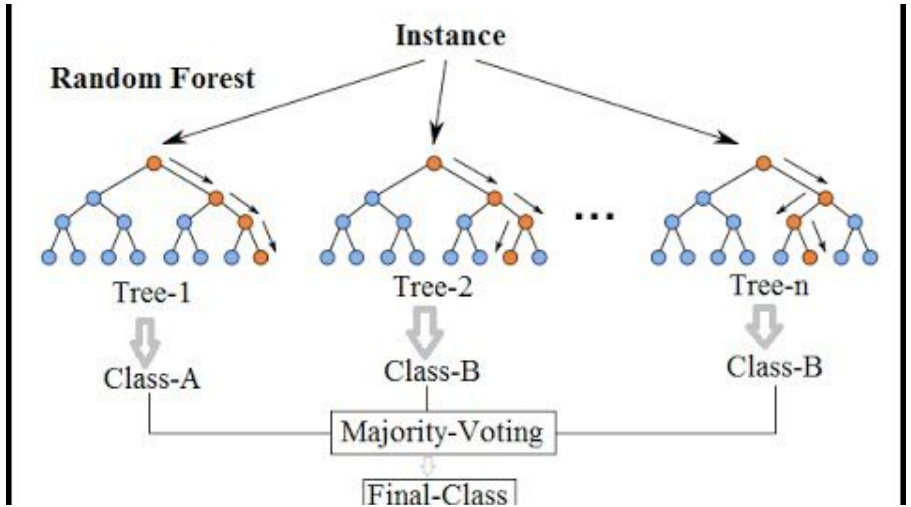
Random Forest

- **Building Decision Trees**: Randomly select samples and features from the original dataset to construct multiple trees.
- **Training Trees**: Each tree is trained independently on a slightly different dataset, using a method known as “Bootstrap aggregating”.
- **Prediction Process**: For classification tasks, the final prediction is made by majority voting from all trees.



Random Forest(cont.)

- **Reduces Overfitting**: By aggregating multiple trees, Random Forest minimizes overfitting to any single dataset.
- **Strong Processing Capability**: Effectively handles large datasets and features without the need for feature selection.
- **Robustness**: Insensitive to outliers and noise, offering good fault tolerance.



Pre-processing steps

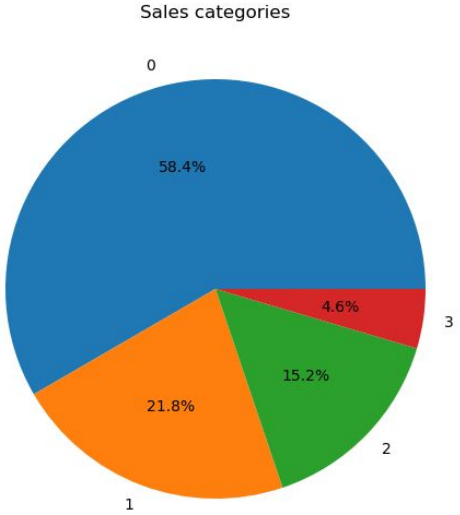
- Restate the initial Prediction problem to an easier-to-handle Classification problem.
- Copying only the features of interest for this task in a new dataframe.
- Compute the conditions under which titles having Global Sales within a certain range will be labelled from 0 to 3.

```
vgClassification = vgsales[['Genre','Platform','Publisher','Year_of_Release','Global_Sales']]
conditions = [
    vgClassification["Global_Sales"] <= 10,
    (vgClassification["Global_Sales"] > 10) & (vgClassification["Global_Sales"] <= 20),
    (vgClassification["Global_Sales"] > 20) & (vgClassification["Global_Sales"] <= 30),
    (vgClassification["Global_Sales"] > 30) & (vgClassification["Global_Sales"] <= 40),
]

values = [0,1,2,3]
vgClassification["Sales_category"] = np.select(conditions, values)
vgClassification.drop("Global_Sales", axis = 1, inplace = True)
```

Pre-processing steps(cont.)

The pieplot below shows the distribution of ***Sales category*** on log scale with the new classes. We notice that the classes remain unbalanced.

```
colors1 = ["red","green","blue",'yellow']
series = vgClassification["Sales_category"].value_counts()
series_sorted = series.sort_index()
fig = plt.figure()
Sales_category = ["0","1","2",'3']
ax = fig.add_axes([0,0,1,1])
ax.pie(np.log(series_sorted),labels = Sales_category,autopct='%1.1f%%')
ax.set_title("Sales categories")
plt.show()
```



Pre-processing steps(cont.)

- Use the Random Over Sampler.
- Randomly duplicate examples in the minority classes so as to distribute them as the samples from the majority class.

```
from collections import Counter
from imblearn.over_sampling import RandomOverSampler

# one-hot encoding
vgClassification = pd.concat([pd.get_dummies(vgClassification[['Genre','Platform','Publisher']]),
 vgClassification[['Year_of_Release','Sales_category']]],axis = 1)

X = vgClassification.drop("Sales_category", axis = 1)
y = vgClassification["Sales_category"]
print('Original dataset shape %s' % Counter(y))
ros = RandomOverSampler(random_state= 42)
X_res, y_res = ros.fit_resample(X, y)
X_new = pd.concat([X_res,y_res],axis = 1)
print('Resampled dataset shape %s' % Counter(y_res))
```

```
Original dataset shape Counter({0: 6786, 1: 27, 2: 10, 3: 2})
Resampled dataset shape Counter({0: 6786, 3: 6786, 2: 6786, 1: 6786})
```

Random Forest Classifier

- The restatement of the problem from regression to classification is much easier to handle.
- The Random Over Sampler balances the categories by resampling video game titles from minority classes.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.33, random_state=42)
model = RandomForestClassifier(n_estimators=10)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(accuracy_score(y_test,y_pred))
```

在 2024.04.12 14:16:55 于 330ms内执行

```
0.9714221924536727
```

Feature Importance

```
featureImp= []
for feat, importance in zip(X_train.columns, model.feature_importances_):
    temp = [feat, importance*100]
    featureImp.append(temp)

fT_df = pd.DataFrame(featureImp, columns = ['Feature', 'Importance'])

idxs = fT_df.sort_values('Importance', ascending = False).index[:10]
print (fT_df.sort_values('Importance', ascending = False))
```

在 2024.04.12 14:16:55 于 12ms内执行

	Feature	Importance
291	Year_of_Release	19.929424
24	Platform_Wii	10.554134
189	Publisher_Nintendo	8.739777
6	Genre_Racing	5.174309
3	Genre_Misc	4.805492
..	...	...
119	Publisher_Graphsim Entertainment	0.000000
193	Publisher_NovaLogic	0.000000
194	Publisher_Number None	0.000000
117	Publisher_Gotham Games	0.000000
71	Publisher_Codemasters Online	0.000000

[292 rows x 2 columns]

Compute the feature importances in terms of mean and standard deviation from the results obtained in each feature tree comprising the forest.

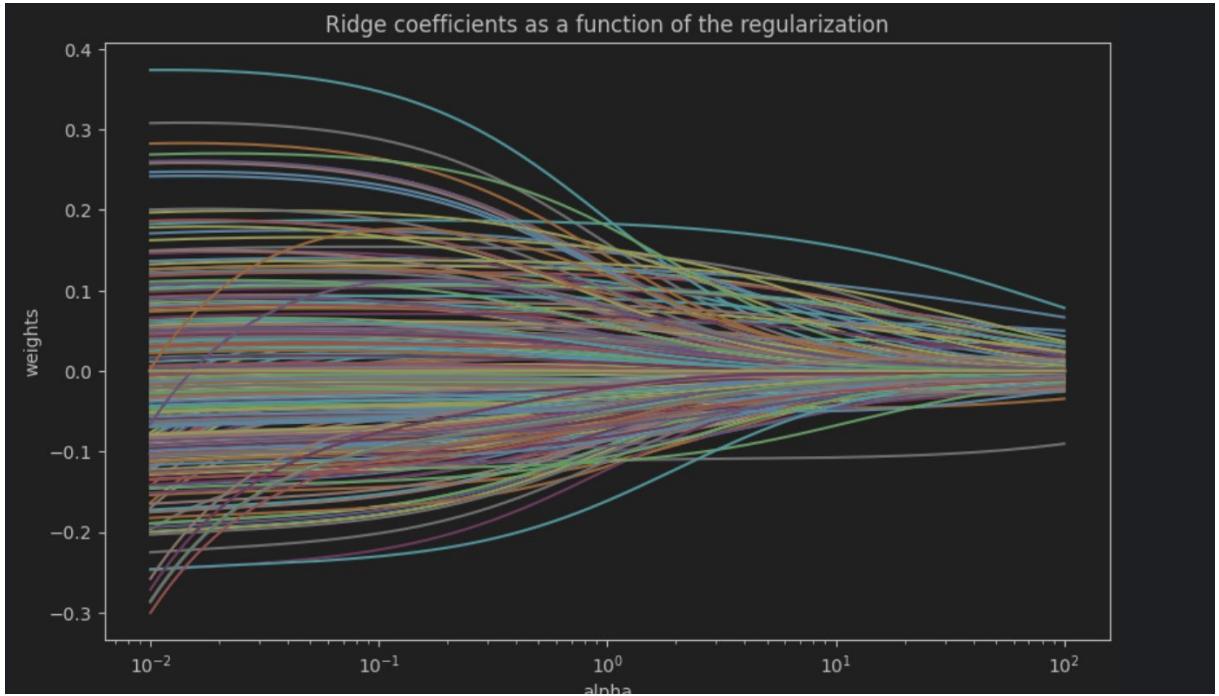
Feature Importance(cont.)

- ***Year*** is by far the most important feature and is a good indicator of the number of Sales for a title.
- The ***Publisher*** Nintendo and its ***Platforms*** Gameboy Advance and Wii have also a remarkable impact for the Sales category.
- Platform, Racing and Misc are the prominent ***genres*** for the classifier.

06

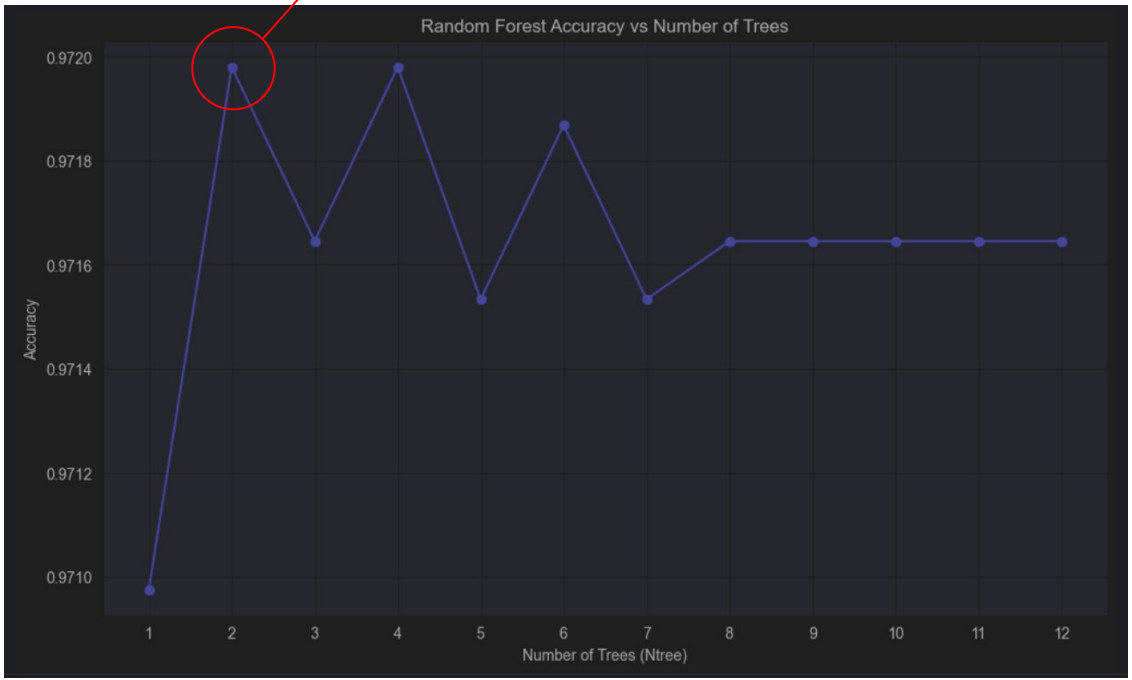
Sensitivity Analysis

Ridge Trace Map



Random Forest

Best: Accuracy Highest with **Number** of Trees is **2**



Other Data Set : Walmart Prediction

Ridge

Ridge Formula: Sum of Error + Sum of the squares of coefficients

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda \sum \beta^2$$

Lasso

Lasso = Sum of Error + Sum of the absolute value of coefficients

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda \sum |\beta|$$

Elastic-Net

Elastic Net Formula: Ridge + Lasso

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda \sum \beta^2 + \lambda \sum |\beta|$$

The Intercept of the Regresion Model was found to be **1047603.298112138**

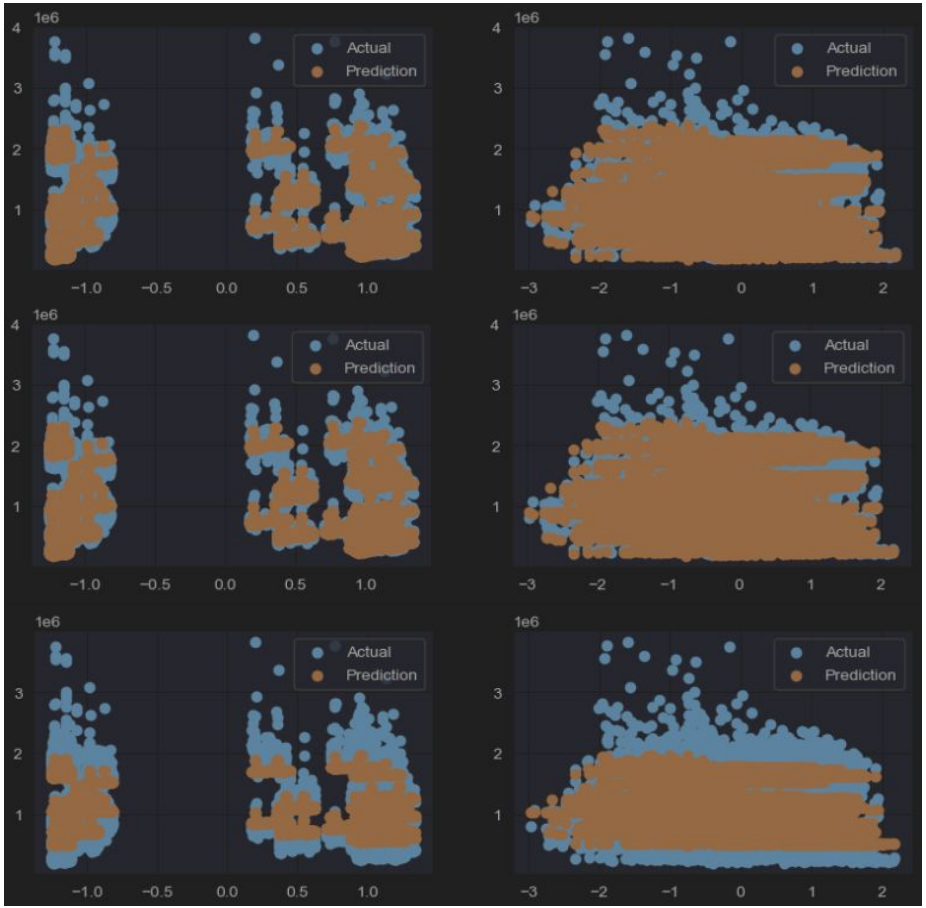
Best alpha parameter found: {'alpha': 3.2374575428176433}
Cross-validation scores: [0.93593586 0.92071592 0.93025392 0.93220418 0.92477374]
Mean cross-validation score: 0.93

Walmart dataset in our model

Ridge Regression

Lasso Regression

Elastic-Net Regression

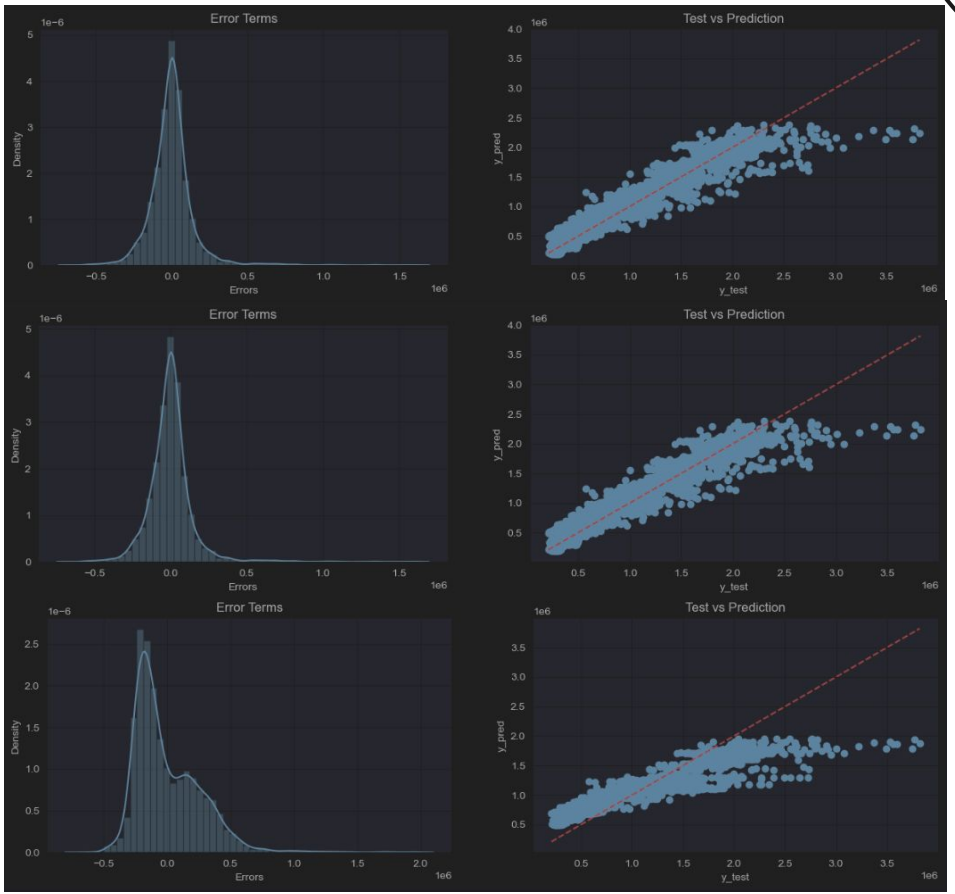


Walmart dataset in our model

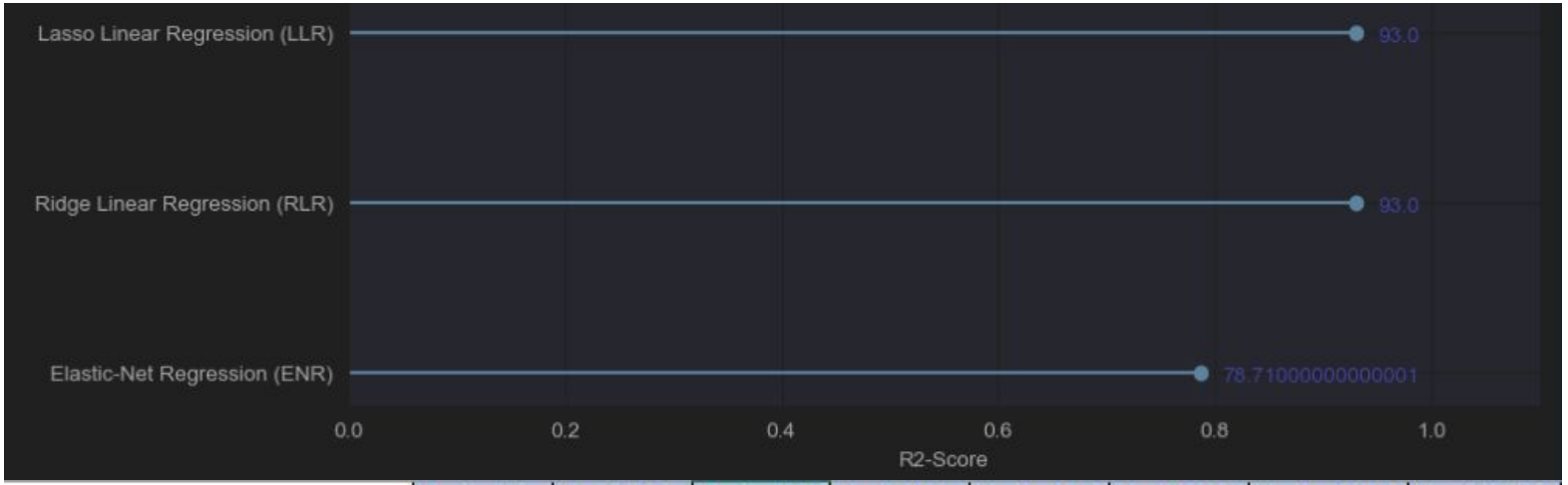
Ridge Regression

Lasso Regression

Elastic-Net Regression



R^2 Scores Comparison for various Regression Models



	Train-R2	Test-R2	Train-RSS	Test-RSS	Train-MSE	Test-MSE	Train-RMSE	Test-RMSE
Ridge Linear Regression (RLR)	0.930048	0.929089	1.08E+14	2.89E+13	2.26E+10	2.43E+10	150461.96	155739.92
Lasso Linear Regression (LLR)	0.93005	0.929058	1.08E+14	2.89E+13	2.26E+10	2.43E+10	150459.66	155773.65
Elastic-Net Regression (ENR)	0.787142	0.792338	3.28E+14	8.46E+13	6.89E+10	7.10E+10	262464.38	266515.5

07

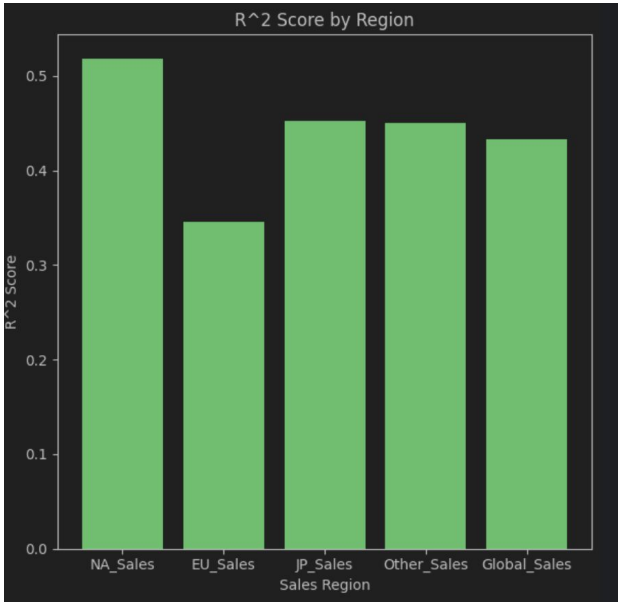
Conclusion

1.Various factors in different regions
2. Publishers are the most important

NA	coefficient
Publisher_Visco	0.229758387
Publisher_1C Company	0.204971822
Publisher_Trion Worlds	0.194105873
Publisher_Sony Computer Entertainment America	0.193951805
Publisher_Square Enix	0.188671535
Publisher_Warner Bros. Interactive Entertainment	0.179299924
Publisher_LucasArts	0.177043433
Publisher_Activision Blizzard	0.166155012
Publisher_SquareSoft	0.165063268

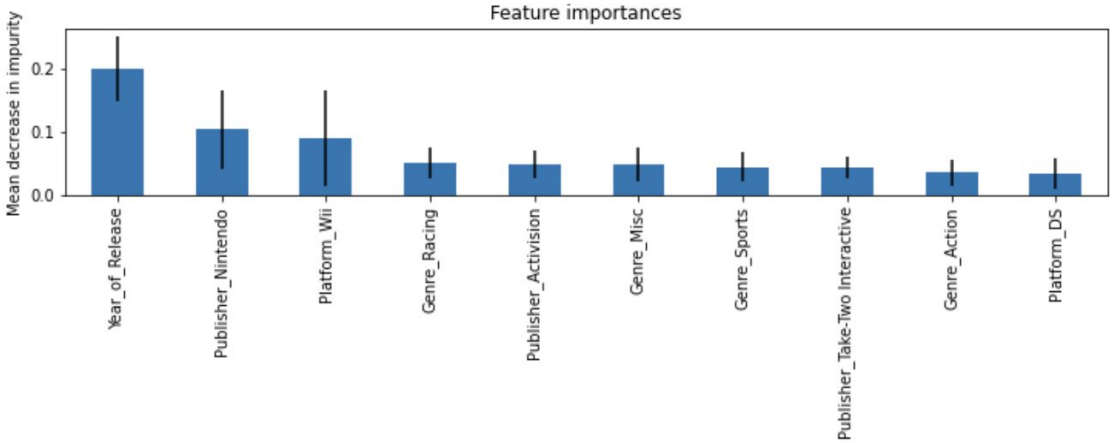
Japan	coefficient
Publisher_Nintendo	0.446345404
Publisher_From Software	0.427927797
Publisher_Ubisoft Annecy	0.419372568
Publisher_Enix Corporation	0.413087685
Publisher_SquareSoft	0.350990548
Publisher_Marvelous Entertainment	0.326428462
Publisher_Gamebridge	0.32191309
Platform_DC	0.3003811
Publisher_Pacific Century Cyber Works	0.290936582
Publisher_Marvelous Interactive	0.283676896

3. Other factors also make sense
4.The factors affecting the world are the most complex



5. Random Forest result best

```
feature_names = [f"{i}" for i in vgClassification.columns[:-1]]
forest_importances = pd.Series(model.feature_importances_, index=feature_names)
best_feat_importances = forest_importances.sort_values(ascending=False)[:10]

std = np.std([tree.feature_importances_ for tree in model.estimators_], axis=0)
best_feat_importances.plot.bar(yerr=std[idxs],figsize = (10,4))
plt.title("Feature importances")
plt.ylabel("Mean decrease in impurity")
plt.show()
```



Reference

[1] Laufer, B., Docherty, P. D., Murray, R., Krueger-Ziolek, S., Jalal, N. A., Hoeflinger, F., Rupitsch, S. J., Reindl, L., & Moeller, K. (2023). Sensor Selection for Tidal Volume Determination via Linear Regression—Impact of Lasso versus Ridge Regression.

[2] SALEH, A. K. Md. E., NAVRÁTIL, R., & NOROUZIRAD, M. (2018). Rank theory approach to ridge, LASSO, preliminary test and Stein-type estimators: A comparative study. *Canadian Journal of Statistics*, *46*(4), 690–704.

[3] Gabauer, D., Gupta, R., Marfatia, H. A., & Miller, S. M. (2024). Estimating U.S. housing price network connectedness: Evidence from dynamic Elastic Net, Lasso, and ridge vector autoregressive models. *International Review of Economics & Finance*, *89*, 349–362.

[4] García-Nieto, P. J., García-Gonzalo, E., & Paredes-Sánchez, J. P. (2021). Prediction of the critical temperature of a superconductor by using the WOA/MARS, Ridge, Lasso and Elastic-net machine learning techniques. *Neural Computing & Applications*, *33*(24), 17131–17145.

[5] Liu, H., & Yu, B. (2013). Asymptotic properties of Lasso+mLS and Lasso+Ridge in sparse high-dimensional linear regression.

THANKS!

Any Questions?