دانشکده مهندسی برق

مینی‌پروژه اول درس هوش مصنوعی

استاد درس:

جناب آقای دکتر علیاری

پریا ساعی ۴۰۱۱۹۱۶۳

ارشیا کلانتریان ۴۰۱۲۱۹۹۳

مریم سلطانی ۴۰۱۱۹۴۳۳

نیمسال دوم ۱۴۰۳_۱۴۰۴

لینک گوگل کولب:

https://colab.research.google.com/drive/1CwmLDhizuF_Ca-QxBUtpzeGJX33euvp4?usp=drive_link

لینک گیت‌هاب:

مریم سلطانی: https://github.com/MaryamSoltani28/AI_2025

ارشیا کلانتریان: https://github.com/ARKAL-J04/MachineLearning2025

پریا ساعی: https://github.com/Paria-s/AI_4032

سوال۱) بخش اول

ابتدا فایل داده‌ها از لینک داده شده دانلود و در گوگل درایو آپلود می‌کنیم سپس آن را در حالت عمومی قرار داده و سپس با کمک قسمت مورد نظر لینک مربوط به فایل (که در اینجا آورده شده)، دیتا ها را با دستور مناسب در گوگل کولب بارگذاری می‌کنیم:

```
#https://drive.google.com/file/d/1nk5uXntvKwh8jU0tkYFFVysNZVJotsyK/view?usp=sharing
!pip install --upgrade --no-cache-dir gdown
!gdown 1nk5uXntvKwh8jU0tkYFFVysNZVJotsyK
```

(۱,۱

در اینجا به کمک متد info. ساختار داده‌ها را شناسایی می‌کنیم. فایل داده شامل ۱۲ ستون است که دو ستون مربوط به سن و کرایه از جنس اعداد اعشاری، ۵ ستون از جنس عدد صحیح و باقی ستون‌ها غیر عددی هستند. بجز ردیف صفرم که Header است، ۸۹۱ ردیف که هرکدام مربوط به یک مسافر است داریم:

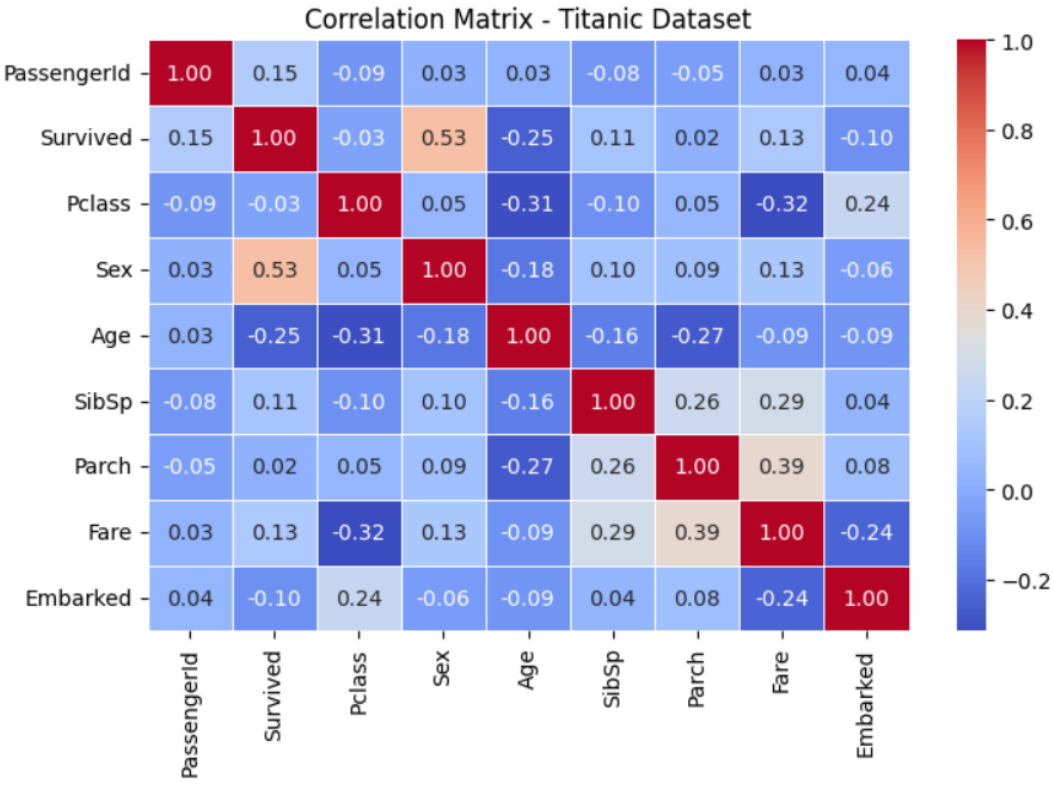
```
# Creating the dataframe
df = pd.read_csv('/content/Titanic-Dataset.csv')

# to print the full summary
print("General information about the dataset:")
df.info()
```

در این بخش ابتدا به داده‌های غیر عددی مثل جنسیت و بندر سوار کشتی شدن، عدد نسبت می‌دهیم سپس با دستور df.dropna() داده‌هایی که مقدار درستی برای آنها وجود ندارد (NaN) را حذف می‌کنیم. داده‌های عددی را در یک متغیر جدید ذخیره می‌کنیم سپس با دستور corr(). از کتابخانه پانداس، ماتریس همبستگی میان آنها را محاسبه کرده(مقادیری بین ۱-و۱ دارد) و در یک متغیر جدید ذخیره می‌کنیم. ماتریس همبستگی را به کمک نمودار حرارتی (رنگ آبی همبستگی زیاد و منفی، رنگ قرمز همبستگی زیاد مثبت و رنگ سفید نشان دهنده خنثی بودن و همبستگی صفر است) نمایش می‌دهیم و در آخر میزان همبستگی هر ویژگی با زنده ماندن را بررسی می‌کنیم:

```
# Convert non-numeric features to numeric
df["Sex"] = df["Sex"].map({"male": 0, "female": 1})  # Male=0, Female=1
df["Embarked"] = df["Embarked"].map({"C": 0, "Q": 1, "S": 2})  # Convert Embarked
values

# Remove missing values (NaN)
df = df.dropna()
```

```
# Select numeric features for correlation calculation
numeric_df = df.select_dtypes(include=['number'])

# Calculate the correlation matrix
corr_matrix = numeric_df.corr()

# Plot the correlation heatmap
plt.figure(figsize=(8, 5))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix - Titanic Dataset")
plt.show()

# Check the correlation of features with the target variable (Survived)
print("Feature Correlation with Survived:\n")
print(corr_matrix["Survived"].sort_values(ascending=False))
```

مشاهده می‌شود که جنسیت بیشترین همبستگی با زنده ماندن را دارد:



مقدار عددی همبستگی بین ویژگی های مختلف و زنده ماندن در اینجا آورده شده است:

```
Feature Correlation with Survived:

Survived       1.000000
Sex            0.532418
PassengerId    0.148495
Fare           0.134241
SibSp          0.106346
Parch          0.023582
Pclass        -0.034542
Embarked      -0.100943
Age           -0.254085
Name: Survived, dtype: float64
```

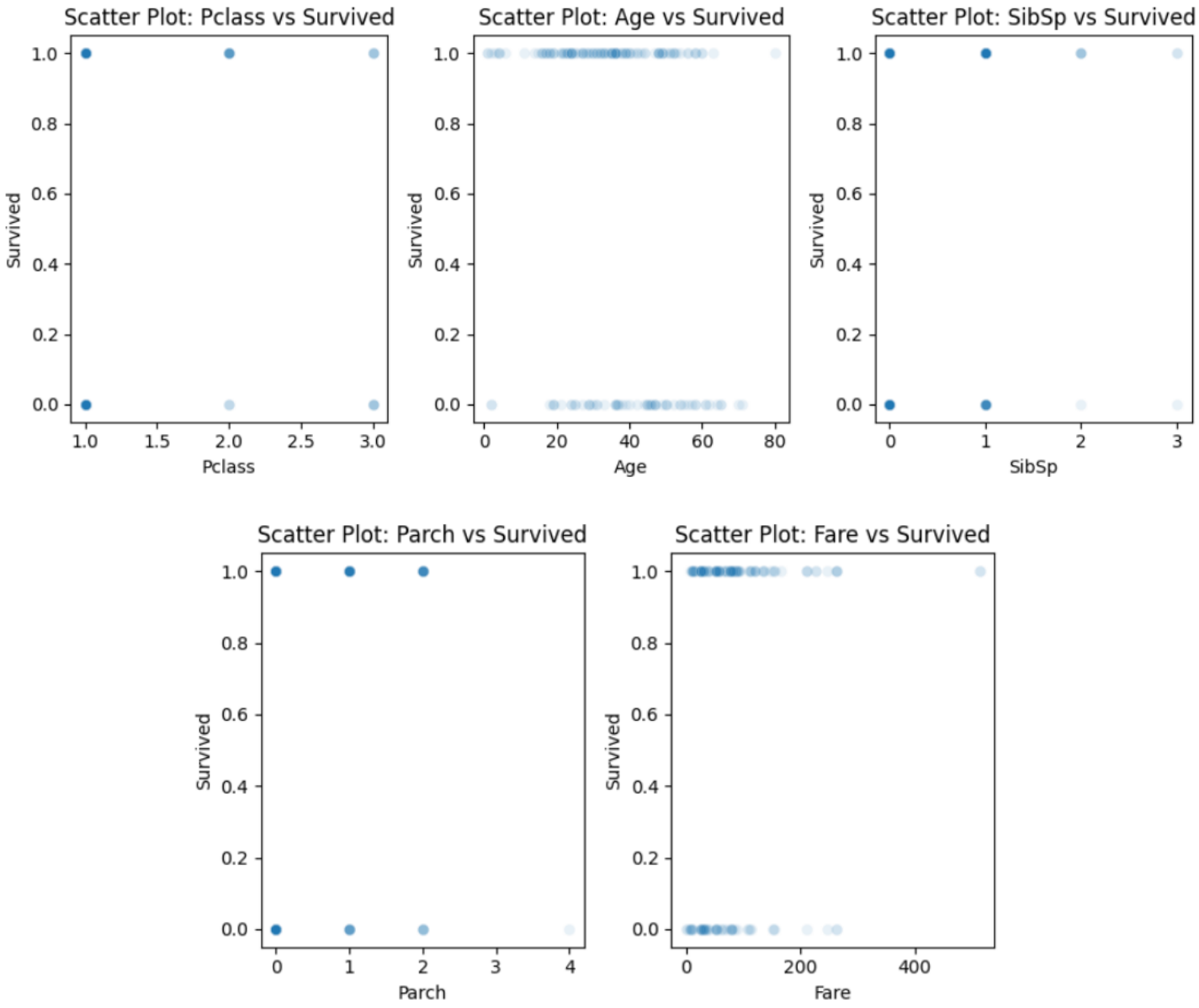
در ابتدا مانند قسمت قبل، داده‌های NaN را حذف می‌کنیم. یک حلقه تشکیل می‌دهیم که ۵ ویژگی مورد نظر را که می‌خواهیم نمودار پراکندگی را برای آنها رسم کنیم، بررسی کرده و برای هر ویژگی یک نمودار ایجاد می‌کند. نمودار پراکندگی را به کمک دستور sns.scatterplot() رسم می‌کنیم.

محور افقی ویژگی مورد نظر و محور عمودی زنده ماندن را که مقدار صفر یا یک می‌گیرد، نمایش می‌دهد. مقدار الفا را که به شفافیت نقاط مربوط می‌شود، ۰,۱ در نظر می‌گیریم تا نقاط واضح بوده و روی هم نیوفتند. plt.tight_layout() فاصله نمودار ها را تنظیم می‌کند.

برای رسم نمودار هگزبین نیز ابتدا به یک حلقه نیاز داریم تا دو ویژگی مورد نظر را بررسی کند و نمودار مربوط به آنها رسم شود. با دستور plt.hexbin() نمودار هگزبین رسم می‌شود.

Gridsize اندازه نقاط داخل نمودار را مشخص می‌کند. Mincnt مشخص می‌کند حداقل مقدار در هر سلول چقدر باید باشد تا در خروجی نمایش داده شود. plt.colorbar(label="Count") مقیاس رنگی و تعداد نقاط مربوط به هر رنگ را مشخص می‌کند.

```
# Drop missing values (NaN) for numeric features
df = df[["Survived", "Pclass", "Age", "SibSp", "Parch", "Fare"]].dropna()

# Plot scatter plots
plt.figure(figsize=(16, 4))
for i, feature in enumerate(["Pclass", "Age", "SibSp", "Parch", "Fare"]):
    plt.subplot(1, 5, i+1)
    sns.scatterplot(data=df, x=feature, y="Survived", alpha=0.1)
    plt.title(f"Scatter Plot: {feature} vs Survived")
plt.tight_layout()
plt.show()

# Plot hexbin plots for continuous features only (age, fare)
plt.figure(figsize=(12, 4))
for i, feature in enumerate(["Age", "Fare"]):
    plt.subplot(1, 2, i+1)
    plt.hexbin(df[feature], df["Survived"], gridsize=25, cmap="coolwarm", mincnt=1)
    plt.colorbar(label="Count")
    plt.xlabel(feature)
    plt.ylabel("Survived")
    plt.title(f"Hexbin Plot: {feature} vs Survived")
plt.tight_layout()
plt.show()
```



تحلیل نمودار:

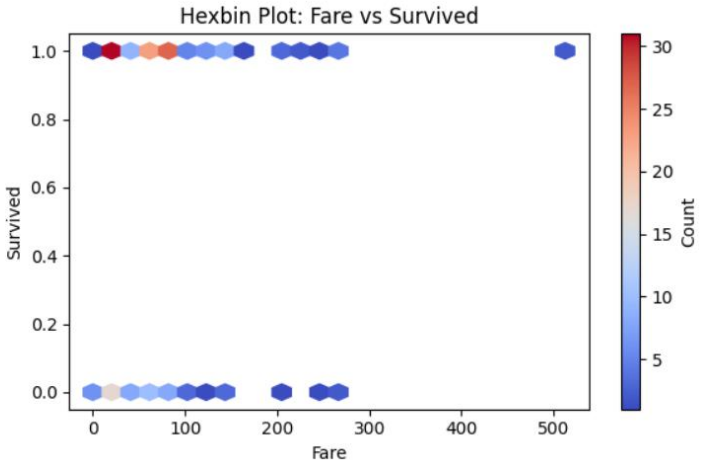
Pclass: احتمال زنده‌ماندن در کلاس‌های بالاتر بیشتر است (کلاس ۱ نجات‌یافته بیشتری دارد).

Age: مسافران جوان‌تر (زیر ۱۵ سال) بیشتر زنده مانده‌اند.

SibSp: مسافران با تعداد زیاد همراه (خانواده) کمتر زنده مانده‌اند.

Parch: افراد با تعداد والدین/فرزندان متوسط (۲-۱) شانس نجات بیشتری داشتند.

Fare: افراد با بلیت‌های گران‌تر بیشتر نجات یافته‌اند.



تحلیل:

Age: جوانان (زیر ۱۵ سال) احتمال بیشتری برای نجات داشته‌اند.

Fare: افراد با بلیت‌های گران‌تر بیشتر زنده مانده‌اند.

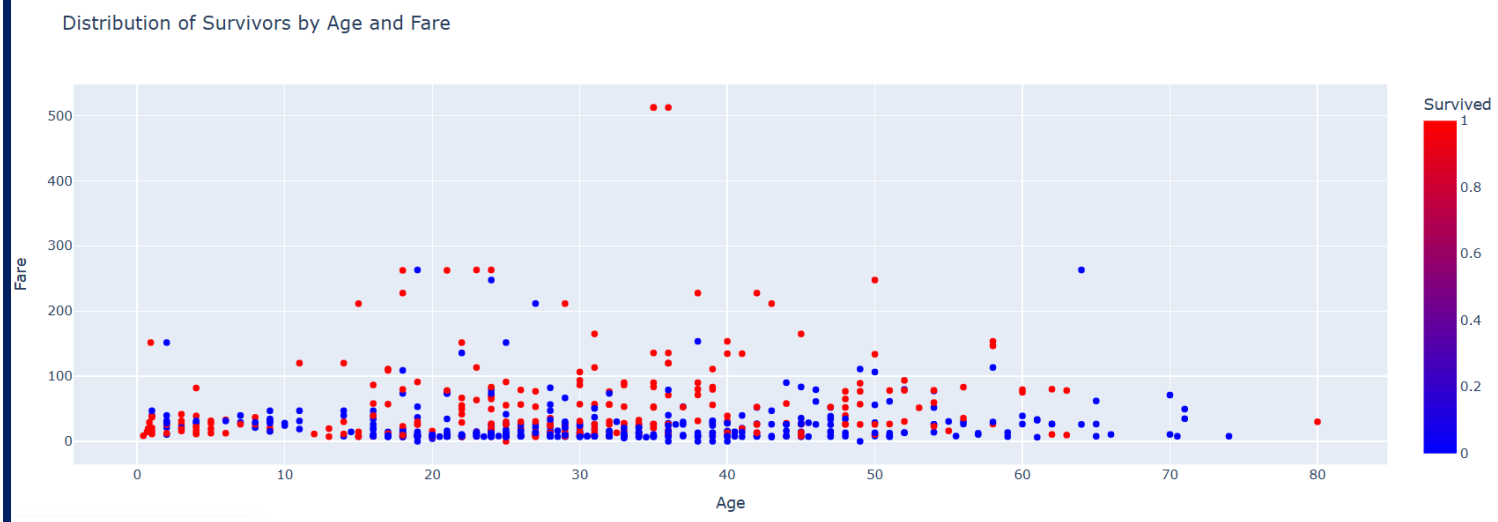
در ادامه نمودار پراکندگی را برای بررسی زنده ماندن بر اساس سن و کرایه پرداخت شده، رسم می‌کنیم. از تابع px.scatter() از کتابخانه Plotly برای رسم نمودار پراکندگی (scatter plot) استفاده می‌کنیم. داده‌ها و دو ویژگی مورد نظر را که همان سن و کرایه است به عنوان ورودی به تابع می‌دهیم همچنین رنگ نمودار را بر اساس زنده ماندن تعریف می‌کنیم.
df.groupby("Survived")داده‌ها را بر اساس ستون Survived (زنده ماندن یا نه) گروه‌بندی می‌کند. و تمرکز گروه‌بندی بر میزان کرایه است.mean()میانگین کرایه بلیط را برای هر گروه (زنده‌مانده‌ها و فوت‌شده‌ها) محاسبه می‌کند. و در ادامه این میانگین در خروجی چاپ می‌شود.

sns.countplot نمودار میله‌ای ایجاد می‌کند که وضعیت زنده ماندن و یا نماندن را بر اساس جنسیت بررسی می‌کند.x=Sex به این معناست که محور افقی نشان دهنده جنسیت است. hue=Survived به این معناست که رنگ میله‌ها بر اساس وضعیت زنده ماندن تنظیم می‌شود.

در آخر نرخ زنده ماندن بر اساس سن(همانند حالتی که زنده ماندن بر اساس کریه بررسی شد) محاسبه شده و نمایش داده می‌شود:

```
df = pd.read_csv("/content/Titanic-Dataset.csv")

# 1. Scatter plot: Distribution of survivors by age and fare
fig = px.scatter(df, x="Age", y="Fare", color="Survived",
                 color_continuous_scale="bluered",  # Change color scheme
                 title="Distribution of Survivors by Age and Fare")
```

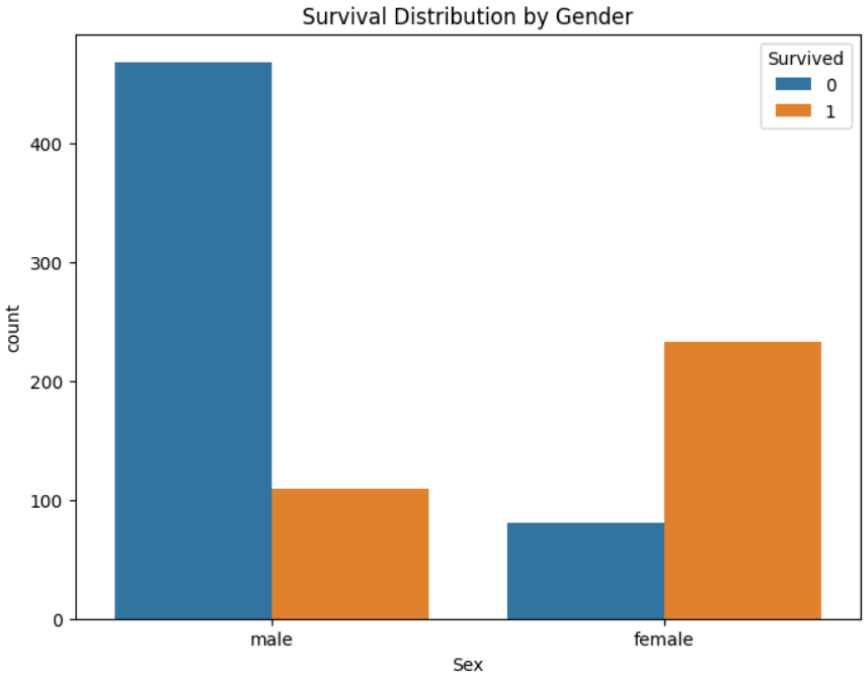
```
fig.show()

# Analyze the relationship between fare and survival
survived_by_fare = df.groupby("Survived")["Fare"].mean()
print("Average fare paid by survivors and non-survivors:")
print(survived_by_fare)

# 2. Count plot: Survival distribution by gender
plt.figure(figsize=(8, 6))
sns.countplot(x="Sex", hue="Survived", data=df)
plt.title("Survival Distribution by Gender")
plt.show()

# Calculate survival rate by gender
survival_rate_by_sex = df.groupby("Sex")["Survived"].mean() * 100
print("Survival rate by gender:")
print(survival_rate_by_sex)
```



مشاهده می‌شود شانس زنده ماندن در افراد با سن کمتر و کرایه پرداختی بیشتر، بیشتر بوده است.

```
Average fare paid by survivors and non-survivors:
Survived
0    22.117887
1    48.395408
Name: Fare, dtype: float64
```

مشاهده می‌شو میانگین کرایه پرداختی در افرادی که زنده مانده‌اند بیشتر بود است.




```
Survival rate by gender:
Sex
female    74.203822
male      18.890815
Name: Survived, dtype: float64
```

با توجه به نمودار و درصد زنده ماندن زنان و مردان، مشخص می‌شود شانس زنده ماندن در زنان بیشتر از مردان بوده است.

(۲,۱)

ابتدا تعداد اعضای خانواده را محاسبه می‌کنیم و خود مسافر را نیز محاسبه می‌کنیم. سپس مسافران تنها را شناسایی می‌کنیم افرادی که تعداد افراد خانواده آنها یک است، تنها محسوب می‌شوند. در اینجا bins = [۰, ۱۲, ۱۸, ۶۰, ۱۰۰] بازه‌های سنی را مشخص می‌کنیم و به آنها لیبل اختصاص می‌دهیم و سپس گروه‌های سنی را تشکیل می‌دهیم.right=False باعث می‌شود که انتهای هر بازه جزو لیبل بازه بعدی قرار بگیرد.

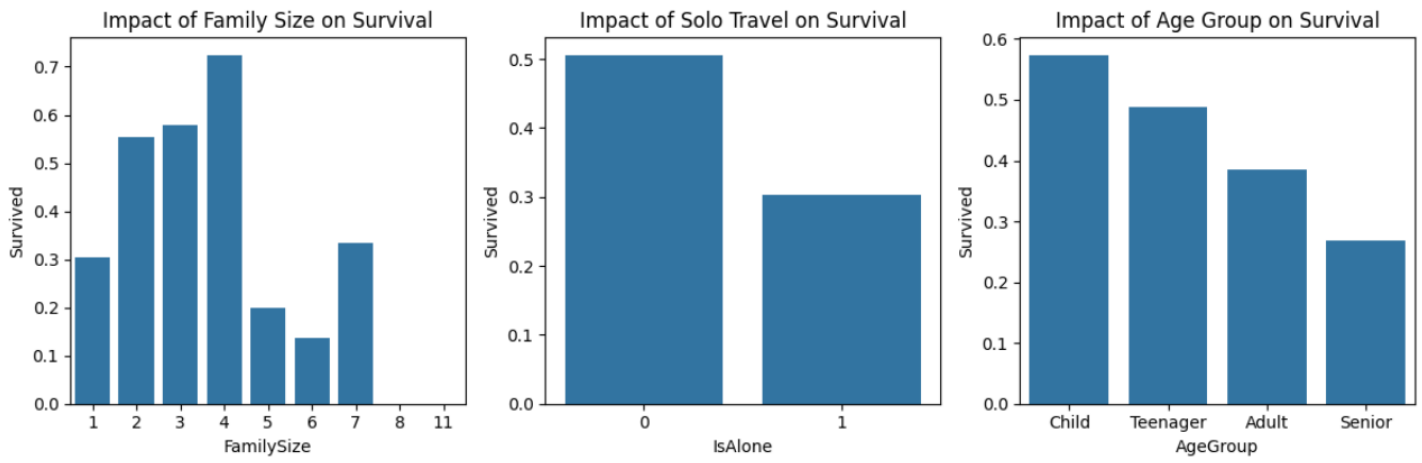
از sns.barplot برای رسم نمودار ستونی استفاده می‌شود و ax=axes[۰] مکان نمودار را مشخص می‌کند.

همانند بخش‌های قبلی به کمک df.groupby تاثیر تعداد اعضای خانواده و گروه سنی را بر زنده ماندن مسافر بررسی می‌کنیم.

IsAlone = 1 نشان دهنده مسافران تنهاست.

```
# 1. Calculate family size and analyze its impact on survival
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1  # Calculate family size

# 2. Analyze solo travel and its impact on survival
df['IsAlone'] = 0
df.loc[df['FamilySize'] == 1, 'IsAlone'] = 1  # Identify solo travelers

# 3. Age group categorization and analysis of its impact on survival
bins = [0, 12, 18, 60, 100]  # Age group bins
labels = ['Child', 'Teenager', 'Adult', 'Senior']
```

```
df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels, right=False)  # Create
age groups

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(12, 4))  # 1 row, 3 columns

# Plot 1: Family size vs. survival (no error bars)
sns.barplot(x='FamilySize', y='Survived', data=df, ax=axes[0], errorbar=None)
axes[0].set_title('Impact of Family Size on Survival')

# Plot 2: Solo travel vs. survival (no error bars)
sns.barplot(x='IsAlone', y='Survived', data=df, ax=axes[1], errorbar=None)
axes[1].set_title('Impact of Solo Travel on Survival')

# Plot 3: Age group vs. survival (no error bars)
sns.barplot(x='AgeGroup', y='Survived', data=df, ax=axes[2], errorbar=None)
axes[2].set_title('Impact of Age Group on Survival')

plt.tight_layout()  # Adjust subplot parameters to give specified padding
plt.show()

# Analyze the relationships
family_survival = df.groupby('FamilySize')['Survived'].mean()
print("Survival rate by family size:")
print(family_survival)

alone_survival = df.groupby('IsAlone')['Survived'].mean()
print("\nSurvival rate for solo and non-solo travelers:")
print(alone_survival)

age_group_survival = df.groupby('AgeGroup', observed=True)['Survived'].mean()
print("\nSurvival rate by age group:")
print(age_group_survival)
```



مشاهده می‌شود شانس بقا در افراد با تعداد اعضای ۴ نفر، و کودکان بیشتر است همچنین افرادی که تنها سفر کرده‌اند شانس بقای کمتری داشته اند.

```
Survival rate by family size:
FamilySize
1     0.303538
2     0.552795
3     0.578431
4     0.724138
5     0.200000
6     0.136364
7     0.333333
8     0.000000
11    0.000000
Name: Survived, dtype: float64

Survival rate for solo and non-solo travelers:
IsAlone
0    0.505650
1    0.303538
Name: Survived, dtype: float64

Survival rate by age group:
AgeGroup
Child       0.573529
Teenager    0.488889
Adult       0.386087
Senior      0.269231
Name: Survived, dtype: float64
```

سوال ۱) بخش دو

در این بخش قصد داریم پیش‌پردازش داده‌ها را انجام دهیم، یعنی آن‌ها را برای انجام رگرسیون لجستیک و ... آماده کنیم.

ممکن است برخی ستون های جدول دارای داده های از دست رفته باشند، تعداد و نسبت این داده ها را به دست بیاورید روش های پر کردن داده های از دست رفته را توضیح دهید و حداقل سه روش را پیاده سازی کنید. دلیل استفاده از هر روش را مختصرا توضیح دهید.

با کد زیر ابتدا داده‌هارا می‌خوانیم و به دیتافریم تبدیل می‌کنیم، سپس تعداد داده‌های گمشده و همچنین درصد هرکدام را نمایش می‌دهیم.

```
df = pd.read_csv('/content/Titanic-Dataset.csv')
# نمایش تعداد داده های گمشده در هر ستون
missing_values = df.isnull().sum()
missing_percentage = (missing_values / len(df)) * 100

# نمایش نتیجه
print(missing_values[missing_values > 0])
print (missing_percentage)
```

```
Age         177
Cabin       687
Embarked      2
dtype: int64
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
```

خروجی به صورت بالا می‌شود. همان‌طور که می‌بینیم تعداد داده گمشده در هر ستون در مقابل نام آن نمایش داده شده است و همچنین درصد آن‌ها را نیز نسبت به کل دادها به دست آوردیم.

روش‌های پر کردن ستون‌ها:

۱ **حذف داده‌ها :** اگر مقدار زیادی از یک ستون گمشده باشد، می‌توانیم کل اون ستون رو حذف کنیم، برای مثال اینجا می‌توانیم ستون cabin را حذف کنیم چون مقدار خیلی زیادی داده از این ستون در دسترس نیست و عملا کاربردی نیست.

۲ **پر کردن با میانگین، میانه یا مد(Mean, Median, Mode)** : اگر مقادیر داده‌های گمشده معقول باید میتوانیم از این روش استفاده کنیم. برای مثال اینجا می‌توانیم برای سلول‌های خالی سن، میانگین سن مسافران را محاسبه کنیم یا برای بندر، از پرتکرارترین بندر (مد) استفاده کنیم.

```
df = df.drop(columns=['Cabin'])
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
```

۳ **پیش‌بینی مقدار گمشده با مدل یادگیری ماشین**

مثلاً، می‌توانیم Age رو با استفاده از ویژگی‌های مرتبط مانند (Pclass, Fare, SibSp, Parch) با کمک مدل رگرسیون یا Random Forest پیش‌بینی کنیم.

آیا امکان حذف برخی ستون ها وجود دارد؟ چرا؟ در صورتی که این امکان وجود دارد با ذکر دلیل ستون های لازم را حذف کنید.

بله همانطور که اشاره شد اگر تعداد داده خیلی زیادی از یک ستون گمشده باشد این کار را می‌کنیم، اینجا ستون cabin را حذف می‌کنیم. و همچنین بعضی ستون‌ها حاوی اطلاعات غیر ضروری هستند مثل:

Name نام مسافر روی بقا تأثیر ندارد.

Ticket شماره بلیت اطلاعات مفیدی ندارد.

PassengerId شناسه‌ی یکتای مسافر، تأثیری روی مدل ندارد.

```
df = df.drop(columns=['PassengerId', 'Name', 'Ticket'])
```

با بررسی دوباره می‌بینیم که دیگر مقدار گمشده نداریم و ستون‌های غیر ضروری نیز حذف شده اند.

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
```

- کدام ویژگی ها را عددی و کدام ها را دسته ای می گویند؟ تفاوت این دو نوع از ویژگی ها در چیست؟ ویژگی های عددی و دسته ای را در این مجموعه دادگان مشخص کنید.

ویژگی عددی(Numerical Features)

ویژگی‌هایی هستند که مقدار آن‌ها عددی بوده و می‌توان آن‌ها را اندازه‌گیری کرد. این ویژگی‌ها معمولاً پیوسته (Continuous)یا گسسته (Discrete) هستند.

مشخصات:

- عملیات ریاضی مثل جمع، میانگین، انحراف معیار و... روی آن‌ها معنی‌دار است.
- اغلب در مدل‌های آماری و ریاضی استفاده می‌شوند.

ویژگی دسته‌ای یا طبقه‌ای(Categorical Features)

ویژگی‌هایی هستند که مقادیر آن‌ها به صورت دسته، گروه یا برچسب هستند و مفهومی عددی ندارند.

مشخصات:

- مقادیر آن‌ها معمولاً متنی هستند یا اگر عددی باشند، جنبه‌ی **معنایی** دارند نه محاسباتی.
- برای استفاده در مدل‌ها معمولاً باید آن‌ها را به کد عددی یا one-hot encoding تبدیل کرد.

در این دیتاست:

Sex مقدارهای "male"و "female"داره.

Embarked مقدارهای "S", "Q", "C"داره.

این داده‌ها از نوع دسته‌ای هستند و باقی آن‌ها از نوع عددی هستند.

مدل یادگیری ماشین **با** متن کار نمی‌کند و فقط عدد می‌فهمد. پس باید این داده‌های متنی رو به عدد تبدیل کنیم.

```
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})

df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)

# تبدیل مقادیر بولی به عددی
df[['Embarked_Q', 'Embarked_S']] = df[['Embarked_Q', 'Embarked_S']].astype(int)

print(df.head())
print(df.dtypes)  # بررسی نوع داده‌ها بعد از تبدیل
```

```
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked_Q  Embarked_S
0         0       3    0  22.0      1      0   7.2500           0           1
1         1       1    1  38.0      1      0  71.2833           0           0
2         1       3    1  26.0      0      0   7.9250           0           1
3         1       1    1  35.0      1      0  53.1000           0           1
4         0       3    0  35.0      0      0   8.0500           0           1
```

```
Survived        int64
Pclass          int64
Sex             int64
Age           float64
SibSp           int64
Parch           int64
Fare          float64
Embarked_Q      int64
Embarked_S      int64
```

همان‌طور که مشاهده می‌کنیم تمامی داده‌ها در نهایت به صورت عددی هستند.

drop_first=True یکی از مقدارها رو حذف می‌کند تا از مشکل همخطی **(Multicollinearity)** جلوگیری شود. حالا به جای Embarked، ستون‌های Embarked_Q و Embarked_S خواهیم داشت (چون Embarked_Cبه‌صورت پیش‌فرض حذف می‌شود.)

نرمال‌سازی ویژگی‌های عددی

چرا؟ بعضی ویژگی‌ها مثل Age و Fare مقیاس‌های خیلی متفاوتی دارند. نرمال‌سازی باعث می‌شود که مدل یادگیری **سریع‌تر و بهتر** آموزش ببیند.

دو روش اصلی وجود دارد:

- استانداردسازی :(Standardization)مقدارها رو طوری تغییر می‌دهد که میانگین = ۰ و انحراف معیار = ۱ بشود.
- نرمال‌سازی :(Min-Max Scaling) مقدارها رو به بازه [0,1]تبدیل می‌کند.

ما در این پروژه از استانداردسازی استفاده می‌کنیم چون برای رگرسیون لجستیک بسیار مناسب‌تر و مفیدتر است. برای این کار ابتدا ا ز کتابخانه sklearn.preprocessin StandardScaler را ایمپورت می‌کنیم و سپس از دستور scaler.fit_transform استفاده می‌کنیم و در نهایت داده‌های پیش پردازش شده را ذخیره می‌کنیم.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[['Age', 'Fare']] = scaler.fit_transform(df[['Age', 'Fare']])
print(df.dtypes)
df.to_csv('cleaned_titanic.csv', index=False)
print(df.head())
```

خروجی به این شکل می‌شود:

```
Survived        int64
Pclass          int64
Sex             int64
Age           float64
SibSp           int64
Parch           int64
Fare          float64
Embarked_Q      int64
```

```
Embarked_S      int64
dtype: object

   Survived  Pclass  Sex       Age  SibSp  Parch      Fare  Embarked_Q  \
0         0       3    0 -0.565736      1      0 -0.502445           0
1         1       1    1  0.663861      1      0  0.786845           0
2         1       3    1 -0.258337      0      0 -0.488854           0
3         1       1    1  0.433312      1      0  0.420730           0
4         0       3    0  0.433312      0      0 -0.486337           0

   Embarked_S
0           1
1           0
2           1
3           1
```

همان‌طور که می‌بینیم داده‌ها کاملا پیش پردازش شده و آماده آموزش به مدل هستند.

سوال ۱) بخش سه

انتخاب ویژگی با استفاده از روش‌های رگرسیون لاسو و انتخاب بازگشتی(RFE)

در این بخش از پروژه، هدف ما انتخاب مهم‌ترین ویژگی‌های مؤثر بر پیش‌بینی بقا در داده‌های تایتانیک است. برای این منظور از دو روش پرکاربرد در حوزه انتخاب ویژگی استفاده کرده‌ایم:

رگرسیون لاسو(Lasso Regression)

انتخاب بازگشتی با حذف ویژگی‌ها(Recursive Feature Elimination - RFE)

۱ .روش رگرسیون لاسو

رگرسیون لاسو نوعی رگرسیون خطی است که از جریمه L1 برای کاهش ضرایب ویژگی‌ها استفاده می‌کند. این جریمه موجب می‌شود برخی ضرایب به صفر میل کنند و در نتیجه آن ویژگی‌ها از مدل حذف شوند. در نتیجه، لاسو علاوه بر تخمین مدل، نقش انتخاب ویژگی را نیز ایفا می‌کند.

ابتدا داده‌ها را به دو مجموعه آموزشی و آزمایشی تقسیم کردیم:

```
# تقسیم داده ها به داده های آموزشی و آزمایشی
X = df2.drop(columns=['Survived'])  # متغیرهای مستقل
y = df2['Survived']  # متغیر هدف

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=63)
```

سپس مدل لاسو با مقدار پارامتر alpha=0.01آموزش داده شد. این پارامتر میزان جریمه را کنترل می‌کند.

```
# ساخت مدل لاسو
lasso = Lasso(alpha=0.01)
lasso.fit(X_train, y_train)
```

ویژگی‌هایی که ضرایب غیر صفر داشتند به عنوان ویژگی‌های انتخاب‌شده در نظر گرفته شدند:

```
# نمایش ویژگی‌های انتخاب‌شده
selected_features_lasso = np.where(lasso.coef_ != 0)[0]
print("Selected features by Lasso:", X.columns[selected_features_lasso])
```

```
Selected features by Lasso: Index(['Pclass', 'Sex', 'Age', 'SibSp', 'Fare', 'Embarked_S'],
dtype='object')
```

. ویژگی‌های انتخاب‌شده توسط لاسو:

Pclass

Sex

Age

SibSp

Fare

Embarked_S

۲ .روش(RFE (Recursive Feature Elimination

روش RFE یک الگوریتم انتخاب ویژگی بازگشتی است که در هر مرحله، ضعیف‌ترین ویژگی را حذف می‌کند و مدل را دوباره آموزش می‌دهد. در نهایت، بهترین مجموعه ویژگی‌ها باقی می‌ماند.

در اینجا، از مدل رگرسیون لجستیک به‌عنوان مدل پایه استفاده کردیم و تعداد ویژگی‌های انتخابی را برابر ۵ قرار دادیم:

```
# ساخت مدل رگرسیون لجستیک
model = LogisticRegression(max_iter=1000)

# پیاده‌سازی RFE
rfe = RFE(model, n_features_to_select=5)  # تعداد ویژگی‌هایی که می‌خواهیم نگه داریم
rfe.fit(X_train, y_train)
```

ویژگی‌های انتخاب‌شده به صورت زیر به دست آمدند:

```
# ویژگی‌های انتخاب‌شده
selected_features_rfe = X.columns[rfe.support_]
print("Selected features by RFE:", selected_features_rfe)
# ویژگی‌های انتخاب‌شده
selected_features_rfe = X.columns[rfe.support_]
```

```
print("Selected features by RFE:", selected_features_rfe)
```

ویژگی‌های انتخاب‌شده توسط RFE :

Pclass

Sex

Age

SibSp

Embarked_S

۳ .ترکیب ویژگی‌های انتخاب‌شده

در نهایت، برای پوشش کامل‌تر، اجتماع ویژگی‌های انتخاب‌شده توسط دو روش را در نظر گرفتیم. به این ترتیب، تمامی ویژگی‌هایی که حداقل توسط یکی از روش‌ها مفید شناخته شده‌اند، انتخاب شدند:

selected_features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Fare', 'Embarked_S']

این مجموعه نهایی از ویژگی‌ها، در ادامه پروژه برای آموزش مدل رگرسیون لجستیک مورد استفاده قرار گرفته است.

مدل‌سازی با استفاده از رگرسیون لجستیک دودویی

پس از انتخاب ویژگی‌های مؤثر با استفاده از روش‌های Lasso و RFE ، در این بخش به آموزش و ارزیابی یک مدل **رگرسیون لجستیک دودویی** برای پیش‌بینی بقای مسافران کشتی تایتانیک می‌پردازیم.

۱ .آماده‌سازی داده‌ها

ابتدا داده‌های تمیز شده (cleaned) بارگذاری شده و ویژگی‌های منتخب برای مدل‌سازی استخراج شدند سپس داده‌ها به نسبت ۸۰٪ برای آموزش و ۲۰٪ برای تست تقسیم شدند:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# خواندن داده های تمیز شده
df2 = pd.read_csv('cleaned_titanic.csv')

# انتخاب ویژگی‌های موردنظر
selected_features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Fare', 'Embarked_S']
X = df2[selected_features].values
```

```
y = df2['Survived'].values  # متغیر هدف

# تقسیم داده ها
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=33)
```

۲ .پیاده‌سازی مدل رگرسیون لجستیک دستی

در این پروژه، برخلاف استفاده از مدل‌های آماده کتابخانه‌ها، مدل رگرسیون لجستیک به صورت دستی پیاده‌سازی شده است. برای این منظور از توابع sigmoid، binary cross-entropy (BCE)، گرادیان و گرادیان نزولی استفاده شده است (مطابق کدهایی که در کلاس حل تمرین یاد گرفتیم.)

حالا به توضیح کد می‌پردازیم:

توابع سیگموید:

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

این تابع تابع سیگموید است که در رگرسیون لجستیک برای پیش‌بینی احتمال استفاده می‌شود. این تابع هر عدد حقیقی را به مقداری بین ۰ و ۱ تبدیل می‌کند.

تابع هزینه (ضرر):Cross-Entropy (BCE)

```
def bce(y, y_hat):
    return -np.mean(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
def gradient(X, y, y_hat):
```

این تابع محاسبه هزینه یا ضرر رگرسیون لجستیک است که با استفاده از **باینری کراس-انتروپی** Binary Cross-Entropy انجام می‌شود.

این تابع تفاوت بین برچسب‌های واقعی (y) و پیش‌بینی‌های مدل (y_hat) را محاسبه می‌کند.

محاسبه گرادیان:

```
def gradient(X, y, y_hat):
    return (X.T @ (y_hat - y)) / len(y)
```

این تابع گرادیان تابع هزینه را محاسبه می‌کند. در رگرسیون لجستیک، گرادیان نشان‌دهنده تغییرات وزن‌ها به ازای تغییرات ورودی‌ها است.

این فرمول برای محاسبه گرادیان است که به ما می‌گوید چگونه باید وزن‌ها را به روز کنیم تا خطای مدل کم شود.

تابع نزول گرادیان:(Gradient Descent)

```
def gradient_descent(w, eta, grads):
    return w - eta * grads
```

این تابع الگوریتم نزول گرادیان را پیاده‌سازی می‌کند. در اینجا، wوزن‌ها، etaنرخ یادگیری و gradsگرادیان‌های محاسبه‌شده هستند.

در هر مرحله از نزول گرادیان، وزن‌ها به اندازه eta * gradsکاهش می‌یابند تا به سمت کمینه تابع هزینه حرکت کنند.

تابع آموزش رگرسیون لجستیک:

```
def train_logistic_regression(X_train, y_train, eta=0.01, epochs=2000):
    m = X_train.shape[1]  # تعداد ویژگی‌ها
    w = np.zeros((m, 1))  # مقداردهی اولیه به وزن‌ها

    y_train = y_train.reshape(-1, 1)

    error_hist = []

    for epoch in range(epochs):
        y_hat = sigmoid(X_train @ w)
        loss = bce(y_train, y_hat)
        error_hist.append(loss)

        grads = gradient(X_train, y_train, y_hat)
        w = gradient_descent(w, eta, grads)

        if (epoch + 1) % 500 == 0:
            print(f'Epoch {epoch+1}: Loss = {loss:.4f}')

    return w, error_hist
```

- این تابع اصلی برای آموزش مدل رگرسیون لجستیک است.
  - ابتدا تعداد ویژگی‌ها (m) و وزن‌ها (w) را مقداردهی اولیه می‌کند. وزن‌ها به صورت یک بردار صفر شروع می‌شوند.
  - سپس داده‌های آموزشی (X_train) و (y_train)آماده می‌شوند.
  - در هر دوره از آموزش که تعداد آن با epochsمشخص شده است:
    - پیش‌بینی‌ها (y_hat) با استفاده از تابع سیگموید محاسبه می‌شوند.

- هزینه یا ضرر با استفاده از تابع bceمحاسبه می‌شود.
- گرادیان‌های مربوط به وزن‌ها با استفاده از تابع gradientمحاسبه می‌شوند.
- وزن‌ها با استفاده از تابع gradient_descentبه‌روزرسانی می‌شوند.
  - در هر ۵۰۰ دوره، مقدار ضرر نمایش داده می‌شود.
- این تابع در نهایت وزن‌های نهایی (w) و تاریخچه خطاها (error_hist) را باز می‌گرداند.

ماتریس ویژگی‌ها با یک ستون بایاس (bias = 1) گسترش داده شد:

```
# اضافه کردن بایاس (ستون ۱ به X)
X_train_bias = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
X_test_bias = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
```

```
# آموزش مدل
w_trained, error_hist = train_logistic_regression(X_train_bias, y_train)
```

در این بخش، مدل رگرسیون لجستیک با داده‌های آموزش (X_train_bias) و آموزش (y_train) داده می‌شود. خروجی این تابع:

- w_trained: وزن‌های نهایی مدل پس از آموزش
- error_hist: تاریخچه خطاها در طول آموزش است که به ما کمک می‌کند تا روند کاهش خطا را مشاهده کنیم.

در حین آموزش، تابع train_logistic_regression خطای مدل را محاسبه کرده و آن را در هر ۵۰۰ دوره (epoch) نمایش می‌دهد.

```
# پیش‌بینی روی داده‌های تستی
y_pred_prob = sigmoid(X_test_bias @ w_trained)
y_pred = (y_pred_prob >= 0.5).astype(int)
```

ابتدا پیش‌بینی‌های احتمال (y_pred_prob) با استفاده از تابع سیگموید و وزن‌های آموزش‌دیده (w_trained) محاسبه می‌شود.

- X_test_bias @ w_trainedیک ضرب ماتریسی است که ورودی‌های تست را با وزن‌ها ضرب می‌کند تا نتایج خطی (نمرات) به دست آید.
- سپس با استفاده از تابع سیگموید، این نمرات به احتمال‌هایی بین ۰ و ۱ تبدیل می‌شوند.

سپس برای تبدیل احتمالات به کلاس‌های ۰ یا ۱، از یک آستانه ۰٫۵ استفاده می‌شود:

- اگر y_pred_prob >= 0.5، به‌عنوان کلاس ۱ (مثبت) پیش‌بینی می‌شود.
- در غیر این صورت، کلاس ۰ (منفی) پیش‌بینی می‌شود.
- astype(int)برای تبدیل نوع داده به عدد صحیح (۰ یا ۱) استفاده می‌شود.

```
# محاسبه دقت
accuracy = np.mean(y_pred.flatten() == y_test)
print(f'Model Accuracy: {accuracy:.2f}')
```

در این بخش، دقت مدل محاسبه می‌شود:

`y_pred.flatten()`برای صاف کردن آرایه پیش‌بینی‌ها به یک آرایه یک‌بعدی استفاده می‌شود.

`y_pred.flatten() == y_test`یک مقایسه منطقی انجام می‌دهد که آیا پیش‌بینی‌ها برابر با برچسب‌های واقعی (`y_test`)هستند یا خیر.

`np.mean()`میانگین این مقایسات منطقی را محاسبه می‌کند، که دقت مدل را به دست می‌دهد.

در نهایت، دقت مدل نمایش داده می‌شود.

```
Epoch 500: Loss = 0.5478
Epoch 1000: Loss = 0.5082
Epoch 1500: Loss = 0.4863
Epoch 2000: Loss = 0.4730

Model Accuracy: 0.80
```

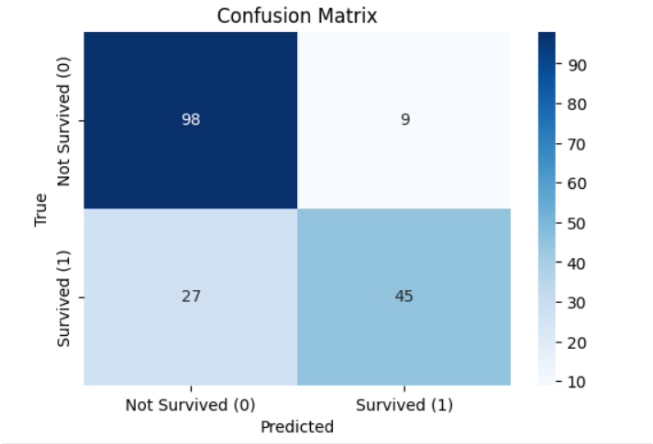
در طول آموزش مدل، مشاهده می‌کنید که مقدار خطا (Loss) در هر ۵۰۰ دوره کاهش می‌یابد. این نشان می‌دهد که مدل به‌طور موثر یاد می‌گیرد و وزن‌ها به‌درستی به‌روزرسانی می‌شوند. در انتها، مدل با دقت ۸۰٪ بر روی داده‌های تست ارزیابی می‌شود که نشان می‌دهد مدل توانسته است ۸۰٪ پیش‌بینی‌های صحیح را انجام دهد.

محاسبه و نمایش ماتریس درهم‌ریختگی:

برای ارزیابی بهتر عملکرد مدل، از ماتریس درهم‌ریختگی (Confusion Matrix) استفاده می‌شود. این ماتریس نشان می‌دهد که مدل چگونه پیش‌بینی‌هایی برای هر کلاس انجام داده است (چند نمونه به درستی پیش‌بینی شده‌اند و چند نمونه اشتباه پیش‌بینی شده‌اند). سپس این ماتریس به‌صورت یک نقشه حرارتی (heatmap) با استفاده از کتابخانه‌های seabornو matplotlibرسم می‌شود.

```
# محاسبه ماتریس درهم‌ریختگی
conf_matrix = confusion_matrix(y_test, y_pred)

# نمایش ماتریس درهم‌ریختگی
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not
Survived (0)', 'Survived (1)'], yticklabels=['Not Survived (0)', 'Survived (1)'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

خروجی به این شکل است:



از این ماتریس می‌توان نتیجه گرفت:

- مدل ۹۸ مورد از افراد نجات‌نیافته را درست پیش‌بینی کرده است.
- مدل ۴۵ مورد از افراد نجات‌یافته را نیز به درستی پیش‌بینی کرده است.
- با این حال، ۲۷مورد از افراد نجات‌یافته به اشتباه نجات‌نیافته تشخیص داده شده‌اند.
- مدل در تشخیص افراد "نجات‌یافته" ضعف بیشتری دارد (کلاس ۱)، که می‌تواند ناشی از عدم توازن داده‌ها یا کم‌اهمیت فرض شدن برخی ویژگی‌های تأثیرگذار باشد.

نمودار ROC و مقدارAUC

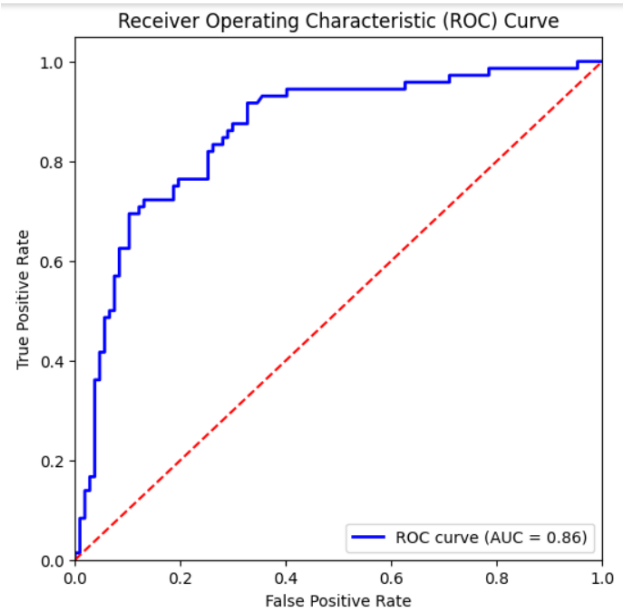
برای ارزیابی جامع‌تر عملکرد مدل رگرسیون لجستیک، از نمودار مشخصه عملکرد گیرنده (ROC) و مساحت زیر منحنی (AUC) استفاده شد.

نمودارROC ، رابطه بین نرخ مثبت واقعی (True Positive Rate) و نرخ مثبت کاذب (False Positive Rate) را برای آستانه‌های مختلف نشان می‌دهد. هر چه این نمودار به سمت گوشه بالا-چپ خمیده‌تر باشد، نشان‌دهنده عملکرد بهتر مدل در تفکیک بین دو کلاس است.

کد مربوط به محاسبه مقادیر FPR و TPR و رسم نمودار ROC در زیر آمده است:

```
from sklearn.metrics import roc_curve, auc

# محاسبه مقادیر FPR و TPR برای رسم ROC
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# رسم نمودار ROC
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# نمایش مقدار AUC
print(f"AUC: {roc_auc:.2f}")
```

این بخش کد مربوط به محاسبه و رسم نمودار ROCو محاسبه AUCاست که برای ارزیابی مدل‌های طبقه‌بندی باینری استفاده می‌شود. ابتدا با استفاده از تابع roc_curveمقادیر نرخ مثبت کاذب (FPR) و نرخ مثبت واقعی (TPR) محاسبه می‌شود که نشان‌دهنده میزان اشتباهات مدل در پیش‌بینی کلاس‌های منفی و مثبت است. سپس با استفاده از این مقادیر، AUCمساحت زیر منحنی ROC محاسبه می‌شود که معیاری برای عملکرد مدل است؛ هرچه AUC نزدیک‌تر به ۱ باشد، مدل عملکرد بهتری دارد. در نهایت نمودار ROCرسم می‌شود که در آن محور افقی نشان‌دهنده FPRو محور عمودی نشان‌دهنده TPRاست. منحنی ROC به بررسی تغییرات نرخ‌های مثبت واقعی و منفی کاذب در آستانه‌های مختلف تصمیم‌گیری پرداخته و مدل را ارزیابی می‌کند. این نمودار به همراه مقدار AUCنمایش داده می‌شود تا میزان کارایی مدل به‌طور تصویری و عددی مشخص شود.



همان‌طور که مشاهده می‌شود، منحنی آبی رنگ (ROC) به‌خوبی از خط قرمز رنگ مرجع (مدل تصادفی) فاصله گرفته است.

مقدار AUC به دست آمده برابر با 0.86است که عدد قابل قبولی محسوب می‌شود. این عدد نشان می‌دهد که مدل توانایی خوبی در تمایز بین دو کلاس "بقا" و "عدم بقا" دارد.

تقسیم داده‌ها به سه دسته

برای دسته‌بندی شانس بقا، از ویژگی Fare استفاده می‌شود که نشان‌دهنده مبلغ پرداختی بلیت است. ابتدا دو آستانه محاسبه می‌شود که به‌وسیله آنها داده‌ها به سه دسته تقسیم می‌شوند. این آستانه‌ها بر اساس درصدهای ۳۳ و ۶۶ از داده‌ها (به‌عبارتی، پایین‌ترین ۳۳ درصد، ۳۳ تا ۶۶ درصد و بیشتر از ۶۶ درصد) تعیین می‌شوند. بنابراین:

- اگر مقدار Fare مسافر کمتر از آستانه اول (۳۳ درصد پایین) باشد، شانس بقا به دسته کم (0)اختصاص می‌یابد.
- اگر مقدار Fare بین این دو آستانه قرار داشته باشد، شانس بقا به دسته متوسط (1)اختصاص می‌یابد.
- اگر مقدار Fare بیشتر از آستانه دوم (۶۶ درصد پایین) باشد، شانس بقا به دسته زیاد (2)اختصاص می‌یابد.

```
# محاسبه آستانه‌های مربوط به Fare
low_threshold = df2['Fare'].quantile(0.33)
medium_threshold = df2['Fare'].quantile(0.66)

# تعریف دسته‌بندی شانس بقا
def classify_survival_chance(row):
    if row['Fare'] <= low_threshold:
        return 0  # کم
    elif row['Fare'] <= medium_threshold:
```

```
        return 1  # متوسط
    else:
        return 2  # زیاد
```

در نهایت، با استفاده از تابعی که این دسته‌بندی را انجام می‌دهد، یک ستون جدید به نام Survival_Chance به داده‌ها اضافه می‌شود که شانس بقا برای هر مسافر را نشان می‌دهد. سپس با استفاده از value_counts، تعداد مسافران در هر دسته (کم، متوسط، زیاد) محاسبه و نمایش داده می‌شود.

```
# اعمال دسته‌بندی روی دیتافریم
df2['Survival_Chance'] = df2.apply(classify_survival_chance, axis=1)
print(df2['Survival_Chance'].value_counts())
```

خروجی به این شکل است:

```
Survival_Chance
1    302
2    295
0    294
```

همانطور که می‌بینیم نتیجه و نحوه تقسیم‌بندی ما از توازن خوبی برخوردار است.

رگرسیون لجستیک چندکلاسه و یکی در مقابل همه

در این بخش قصد داریم رگرسیون لجستیک چند کلاسه انجام دهیم. کد این بخش به شرح زیر می‌باشد و سپس به توضیح آن می‌پردازیم.

```
selected_features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Fare', 'Embarked_S']
X = df2[selected_features]
y = df2['Survival_Chance']  # استفاده از دسته‌بندی جدید

# تقسیم داده‌ها با stratify
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,
random_state=42)
# مدل رگرسیون لجستیک چندکلاسه (Softmax)
model_multi = LogisticRegression(multi_class='multinomial', solver='lbfgs',
max_iter=1000)
model_multi.fit(X_train, y_train)

y_pred_multi = model_multi.predict(X_test)
accuracy_multi = accuracy_score(y_test, y_pred_multi)
print(f"Accuracy (Multinomial): {accuracy_multi:.4f}")

# رسم ماتریس درهم‌ریختگی
conf_matrix_multi = confusion_matrix(y_test, y_pred_multi)
plt.figure(figsize=(6,4))
```

```
sns.heatmap(conf_matrix_multi, annot=True, fmt='d', cmap='Blues', xticklabels=['Low',
'Medium', 'High'], yticklabels=['Low', 'Medium', 'High'])
plt.title('Confusion Matrix - Multinomial')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

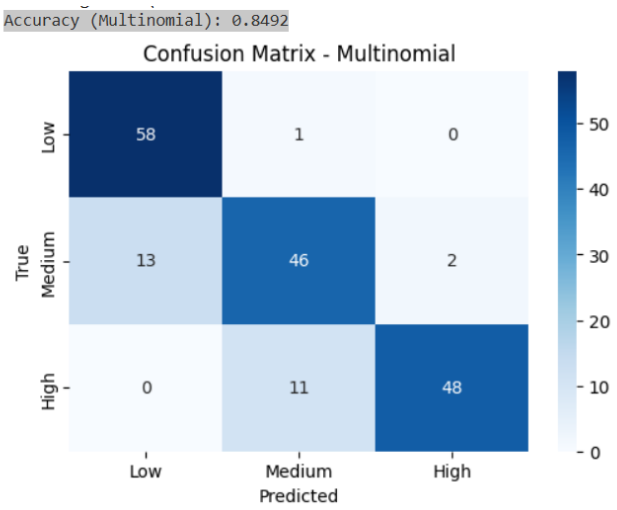
متغیر هدف، یعنی Survival_Chance(دسته‌بندی عددی مربوط به شانس بقا: کم، متوسط، زیاد)، از داده‌ها جدا می‌شود و با استفاده از train_test_splitداده‌ها به دو بخش آموزش و تست تقسیم می‌شوند. پارامتر stratify=y باعث می‌شود که نسبت کلاس‌ها در هر دو مجموعه آموزش و تست ثابت بماند.

در مرحله بعد، یک مدل رگرسیون لجستیک چندکلاسه با استفاده از پارامتر multi_class='multinomial'و حل‌کننده lbfgsساخته و آموزش داده می‌شود. این مدل از نسخه‌ی Softmax برای دسته‌بندی سه‌کلاسه استفاده می‌کند.

پس از آموزش، مدل برای پیش‌بینی دسته‌ی شانس بقا روی داده‌های تست استفاده می‌شود و دقت مدل محاسبه و چاپ می‌گردد.

در انتها، یک ماتریس درهم‌ریختگی رسم می‌شود تا ببینیم مدل چه‌قدر خوب توانسته هر دسته را پیش‌بینی کند. سطرهای این ماتریس نشان‌دهنده مقادیر واقعی و ستون‌ها پیش‌بینی مدل هستند. خانه‌های قطر اصلی نشان می‌دهند که مدل در پیش‌بینی آن دسته درست عمل کرده، و خانه‌های بیرون از قطر اصلی خطاهای مدل را نشان می‌دهند. رنگ آبی ماتریس نشان‌دهنده شدت فراوانی پیش‌بینی‌ها در هر بخش است. این ماتریس به درک بهتر عملکرد مدل در هر کلاس کمک می‌کند.

خروجی به شکل زیر می‌شود:



همان‌طور که می‌بینیم مدل به خوبی عمل کرده است و همچنین مقدار دقت خوبی دارد. حالا به روش یکی در مقابل همه می‌پردازیم:

در این بخش از کد، از روش One-vs-Restیک در برابر بقیه برای پیاده‌سازی رگرسیون لجستیک چندکلاسه استفاده شده است. برخلاف روش قبلی که از مدل multinomial (Softmax) استفاده می‌کرد، این روش هر کلاس را جداگانه در مقابل سایر کلاس‌ها مدل می‌کند، یعنی سه مدل باینری مجزا برای دسته‌های "کم"، "متوسط" و "زیاد" ساخته می‌شود. در ادامه مراحل کار را کامل توضیح می‌دهیم:

```
# مدل One-vs-Rest
base_model = LogisticRegression(solver='liblinear', max_iter=1000)
model_ovr = OneVsRestClassifier(base_model)
model_ovr.fit(X_train, y_train)

y_pred_ovr = model_ovr.predict(X_test)
accuracy_ovr = accuracy_score(y_test, y_pred_ovr)
print(f"Accuracy (One-vs-Rest): {accuracy_ovr:.4f}")

# رسم ماتریس درهم‌ریختگی
conf_matrix_ovr = confusion_matrix(y_test, y_pred_ovr)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_ovr, annot=True, fmt='d', cmap='Oranges', xticklabels=['Low',
'Medium', 'High'], yticklabels=['Low', 'Medium', 'High'])
plt.title('Confusion Matrix - One-vs-Rest')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

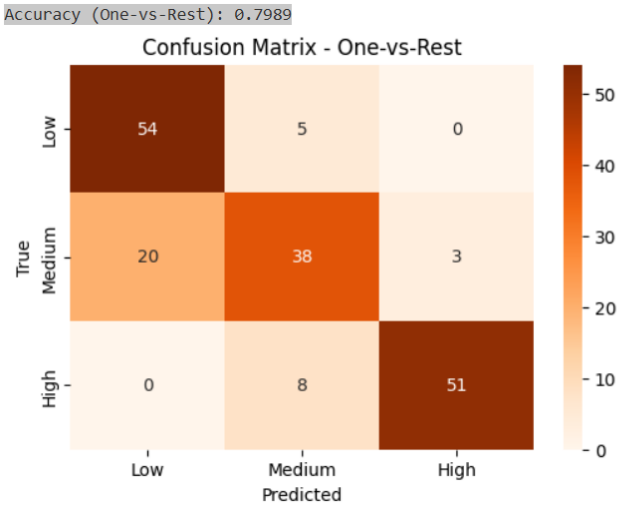
ابتدا یک مدل پایه رگرسیون لجستیک تعریف می‌شود با استفاده از حل‌کننده liblinearکه برای مسائل باینری مناسب است. سپس این مدل در داخل یک کلاس‌بند One-vs-Rest قرار می‌گیرد که خودش به‌طور خودکار سه مدل جداگانه می‌سازد (برای هر کلاس یکی).

مدل روی داده‌های آموزشی آموزش داده می‌شود و سپس بر روی داده‌های تست اعمال می‌شود. خروجی‌های پیش‌بینی شده با مقدار واقعی مقایسه می‌شوند و دقت مدل محاسبه می‌شود. این دقت نشان می‌دهد که چند درصد از پیش‌بینی‌ها به‌درستی انجام شده‌اند.

در پایان، ماتریس درهم‌ریختگی (Confusion Matrix) برای این مدل نیز رسم می‌شود. این ماتریس به ما نشان می‌دهد که مدل در تشخیص هر دستهLow ، Medium، Highچه‌قدر خوب عمل کرده و در کجاها اشتباه داشته. هرچه مقدارهای قطر

اصلی ماتریس بیشتر باشند، یعنی مدل بهتر در تشخیص درست عمل کرده. رنگ‌های استفاده‌شده (در اینجا نارنجی) شدت درستی یا نادرستی را نشان می‌دهند.

در مجموع، این مدل جایگزینی برای مدل Softmax است و با مقایسه دقت و ماتریس‌های درهم‌ریختگی هر دو مدل، می‌توان تصمیم گرفت کدام روش بهتر عمل کرده است.



همانطور که میبینیم در مقایسه با روش لجستیک چند کلاسه، این روش عملکرد ضعیف‌تری داشته. پس نتیجه می‌گیریم بهتر است از روش رگرسیون لجستیک چندکلاسه استفاده کنیم.

ضرایب رگرسیون لجستیک دودویی و چندکلاسه

در این بخش قصد داریم ضرایب و نسبت شانس را برای ویژگی‌ها حساب کنیم و ببینیم هر کدام چگونه تاثیر می‌گذارند.

ضریب (Coefficient) در مدل‌های رگرسیون لجستیک نشان‌دهنده تغییرات پیش‌بینی‌شده در لاجیت (log-odds) احتمال وقوع یک کلاس خاص به ازای هر واحد تغییر در ویژگی مربوطه است.

نسبت شانس (Odds Ratio) که با استفاده از ضریب محاسبه می‌شود(e^coefficient) ، نشان‌دهنده تغییر در احتمال وقوع یک رویداد به ازای تغییر یک واحد در ویژگی مربوطه است.

ابتدا برای لجستیک دودویی انجام می‌دهیم:

```
import numpy as np
import pandas as pd

# در selected_features و w_trained فرض بر اینکه مدل باینری قبلاً آموزش داده شده و دسترس هستند
```

```
# حذف بایاس برای گزارش ضرایب مربوط به ویژگی‌ها
coef_binary = w_trained[1:].flatten()  # چون w_trained شامل بایاس است، از ایندکس ۱ به بعد برداشته میشه
features = selected_features

# محاسبه نسبت شانس (Odds Ratio)
odds_ratios_binary = np.exp(coef_binary)

# ساخت DataFrame برای نمایش
binary_results = pd.DataFrame({
    'Feature': features,
    'Coefficient': coef_binary,
    'Odds Ratio': odds_ratios_binary
})

# تفسیر ویژگی‌ها
binary_results['Interpretation'] = binary_results['Coefficient'].apply(
    lambda x: 'Increases survival probability' if x > 0 else 'Decreases survival probability'
)

print("Binary Logistic Regression Coefficients:\n")
print(binary_results)
```

توضیح کد:

در این بخش، هدف تحلیل ضرایب مدل رگرسیون لجستیک باینری است که قبلاً آموزش داده شده. تمرکز اصلی این تحلیل بر تفسیر ضرایب مدل و درک تأثیر هر ویژگی بر احتمال زنده ماندن (survival probability) است. توضیح کامل این فرآیند به‌صورت زیر است:

ابتدا، چون وزن‌ها (ضرایب) شامل مقدار بایاس (intercept) نیز هستند، فقط ضرایب مربوط به ویژگی‌ها در نظر گرفته می‌شود، یعنی از ایندکس ۱ به بعد. این ضرایب با ترتیب ویژگی‌هایی که قبلاً در مدل استفاده شده‌اند (selected_features) هماهنگ هستند.

در مرحله بعد، با استفاده از تابع نمایی(exp) ، نسبت شانس یا Odds Ratioبرای هر ویژگی محاسبه می‌شود. این مقدار نشان می‌دهد که با یک واحد افزایش در مقدار ویژگی، چقدر احتمال زنده ماندن تغییر می‌کند:

- اگر مقدار بیشتر از ۱ باشد، یعنی آن ویژگی باعث افزایش احتمال بقا می‌شود.
- اگر کمتر از ۱ باشد، یعنی آن ویژگی باعث کاهش احتمال بقا می‌شود.

برای نمایش و تفسیر بهتر، این مقادیر در یک جدول قرار می‌گیرند که شامل نام ویژگی، ضریب خام، نسبت شانس، و تفسیر کیفی (افزایش یا کاهش احتمال بقا) است.

خروجی به صورت زیر است:

```
Binary Logistic Regression Coefficients:
```

	Feature	Coefficient	Odds Ratio	Interpretation
0	Pclass	-0.483896	0.616377	Decreases survival probability
1	Sex	1.476250	4.376501	Increases survival probability
2	Age	-0.370122	0.690650	Decreases survival probability
3	SibSp	-0.184038	0.831904	Decreases survival probability
4	Fare	0.476465	1.610371	Increases survival probability
5	Embarked_S	-0.149676	0.860987	Decreases survival probability

تفسیر خروجی:

جنسیت (Sex) قوی‌ترین عامل در پیش‌بینی بقا بوده است؛ به طوری که با ضریب مثبت ۱٫۴۷ و نسبت شانس ۴٫۳۸، نشان می‌دهد که احتمال زنده ماندن برای زنان حدود ۴٫۴ برابر بیشتر از مردان است.

کلاس بلیت (Pclass) نیز نقش مهمی دارد. ضریب منفی آن (−۰٫۴۸) و نسبت شانس ۰٫۶۱ نشان می‌دهد که مسافران کلاس‌های پایین‌تر (به‌ویژه کلاس ۳) نسبت به مسافران کلاس ۱ شانس بقای کمتری داشته‌اند. به‌احتمال زیاد، این موضوع به محل قرارگیری کابین‌ها و دسترسی دشوارتر به قایق‌های نجات مرتبط بوده است.

سن (Age) نیز با ضریب منفی (−۰٫۳۷) بر کاهش احتمال بقا تأثیرگذار بوده است. هرچه سن بالاتر می‌رفته، شانس زنده ماندن کمتر می‌شده است. این موضوع می‌تواند ناشی از اولویت دادن به کودکان در عملیات نجات باشد.

کرایه بلیت (Fare) با ضریب مثبت ۰٫۴۷ و نسبت شانس ۱٫۶۱ نشان می‌دهد که پرداخت مبلغ بالاتر برای بلیت احتمالاً با کلاس بالاتر سفر و در نتیجه شانس بالاتر بقا همراه بوده است.

تعداد خواهر/برادر یا همسر همراه (SibSp) با ضریب منفی خفیف (−۰٫۱۸) اثر کاهنده بر بقا داشته است. یکی از دلایل ممکن این است که افرادی که با خانواده سفر می‌کردند، ممکن بود در لحظه‌های بحرانی تصمیم‌گیری کندتر عمل کرده یا اولویت نجات اعضای خانواده را در نظر گرفته باشند.

در نهایت، ویژگی (**سوار شدن از بندر ساوت‌همپتون**) نیز با ضریب −۰٫۱۵ و نسبت شانس ۰٫۸۶ نشان می‌دهد که احتمال بقا برای این گروه کمی کمتر از مسافرانی بوده که از سایر بنادر سوار شده‌اند، که می‌تواند به عوامل مختلفی مانند توزیع جمعیتی، محل کابین‌ها یا تعداد مسافران کلاس پایین‌تر مرتبط باشد.

حالا برای رگرسیون لجستیک چندکلاسه این کار را می‌کنیم:

```python
import pandas as pd
import numpy as np

# فرض بر اینکه model_multi همان مدل آموزش‌دیده شده است
# و selected_features نیز در دسترس است
```

```
coef_multi = model_multi.coef_  # شکل: (تعداد کلاس‌ها، تعداد ویژگی‌ها)
classes = model_multi.classes_

for i, class_label in enumerate(classes):
    print(f"\n کلاس{class_label} ({'Low' if class_label == 0 else 'Medium' if
class_label == 1 else 'High'})")

    coefs = coef_multi[i]
    odds_ratios = np.exp(coefs)

    class_results = pd.DataFrame({
        'ویژگی (Feature)': selected_features,
        'ضریب (Coefficient)': np.round(coefs, 4),
        'نسبت شانس (Odds Ratio)': np.round(odds_ratios, 4),
        'تفسیر (Interpretation)': ['🔺 افزایش احتمال' if c > 0 else '🔻 کاهش احتمال'
for c in coefs]
    })

    print(class_results.to_string(index=False))
```

در این بخش از کد، هدف تحلیل ضرایب مدل رگرسیون لجستیک چندکلاسه (multinomial logistic regression) است. ابتدا از مدل آموزش‌دیده‌شده (model_multi) ضرایب هر کلاس استخراج می‌شود. این ضرایب در coef_ذخیره شده‌اند که یک آرایه دوبعدی با شکل (تعداد کلاس‌ها، تعداد ویژگی‌ها) است؛ یعنی هر ردیف نشان‌دهنده ضرایب مربوط به یک کلاس خاص (مثلاً Low یا Medium یا High) است. همچنین کلاس‌های موجود در مدل با استفاده از classes_گرفته می‌شوند که معمولاً شامل مقادیر ۰، ۱ و ۲ هستند (که در این پروژه به ترتیب نمایانگر "شانس کم"، "متوسط" و "زیاد" برای بقا هستند).

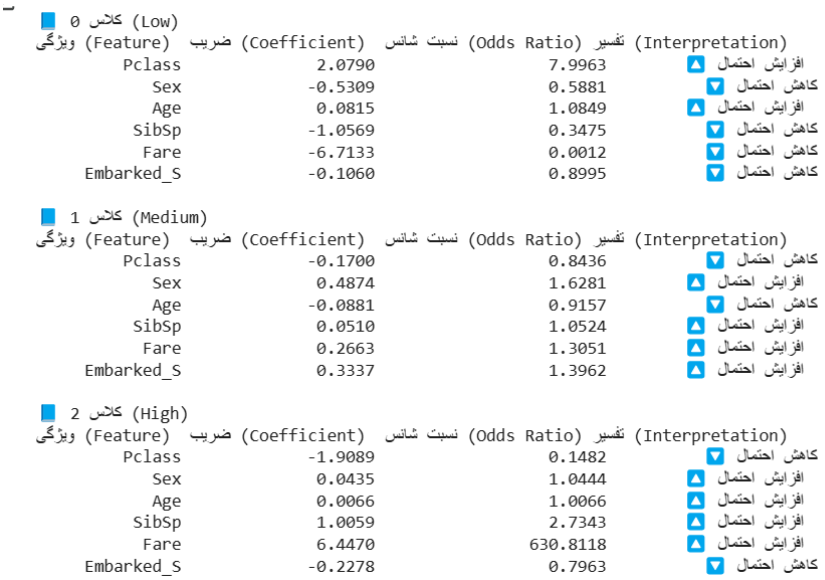
سپس با یک حلقه روی کلاس‌ها پیمایش می‌شود. برای هر کلاس، ضرایب مربوط به آن کلاس جدا می‌شود و با استفاده از تابع نمایی (exp) به نسبت شانس (Odds Ratio) تبدیل می‌گردد. نسبت شانس به ما می‌گوید که افزایش یک واحد در آن ویژگی، احتمال تعلق به آن کلاس خاص را چقدر افزایش یا کاهش می‌دهد.

در ادامه، یک DataFrame ساخته می‌شود که شامل چهار ستون است: نام ویژگی، ضریب آن ویژگی برای کلاس فعلی، نسبت شانس با اعمال exp ، و یک تفسیر ساده از اینکه آیا این ویژگی باعث افزایش یا کاهش احتمال تعلق داده به آن کلاس می‌شود. این تفسیر به صورت نمادهای 🔺 یا 🔻 نشان داده شده است که به ترتیب به معنی "افزایش احتمال" یا "کاهش احتمال" است.

در نهایت، برای هر کلاس این جدول چاپ می‌شود تا بتوان تحلیل دقیقی از نقش ویژگی‌ها در پیش‌بینی آن کلاس داشت. این تحلیل به ما کمک می‌کند بفهمیم کدام ویژگی‌ها برای پیش‌بینی افراد با شانس کم یا زیاد بقا تأثیرگذارتر هستند و جهت تأثیر

آن‌ها چیست (مثبت یا منفی). به عنوان مثال اگر ضریب ویژگی "Fare" برای کلاس High بسیار زیاد باشد و نسبت شانس خیلی بالا باشد، یعنی افزایش کرایه پرداختی به‌شدت احتمال تعلق فرد به کلاس با شانس بقای زیاد را بالا می‌برد.

حال نتایج را بررسی می‌کنیم:



کلاس 0 (Low)

ویژگی (Feature)	ضریب (Coefficient)	نسبت شانس (Odds Ratio)	تفسیر (Interpretation)
Pclass	2.0790	7.9963	افزایش احتمال
Sex	-0.5309	0.5881	کاهش احتمال
Age	0.0815	1.0849	افزایش احتمال
SibSp	-1.0569	0.3475	کاهش احتمال
Fare	-6.7133	0.0012	کاهش احتمال
Embarked_S	-0.1060	0.8995	کاهش احتمال

کلاس 1 (Medium)

ویژگی (Feature)	ضریب (Coefficient)	نسبت شانس (Odds Ratio)	تفسیر (Interpretation)
Pclass	-0.1700	0.8436	کاهش احتمال
Sex	0.4874	1.6281	افزایش احتمال
Age	-0.0881	0.9157	کاهش احتمال
SibSp	0.0510	1.0524	افزایش احتمال
Fare	0.2663	1.3051	افزایش احتمال
Embarked_S	0.3337	1.3962	افزایش احتمال

کلاس 2 (High)

ویژگی (Feature)	ضریب (Coefficient)	نسبت شانس (Odds Ratio)	تفسیر (Interpretation)
Pclass	-1.9089	0.1482	کاهش احتمال
Sex	0.0435	1.0444	افزایش احتمال
Age	0.0066	1.0066	افزایش احتمال
SibSp	1.0059	2.7343	افزایش احتمال
Fare	6.4470	630.8118	افزایش احتمال
Embarked_S	-0.2278	0.7963	کاهش احتمال

تحلیل نتایج

کلاس ۰(Low)

- Pclass (2.0790, Odds Ratio: 7.9963): ضریب مثبت نشان می‌دهد که با افزایش کلاس بلیط (یعنی رفتن به کلاس‌های بالاتر)، احتمال تعلق به کلاس ۰ (شانس بقا کم) افزایش می‌یابد. نسبت شانس بیشتر از ۱ نشان‌دهنده این است که با بالاتر رفتن کلاس بلیط، احتمال بقا در این کلاس افزایش می‌یابد.
- Sex (-0.5309, Odds Ratio: 0.5881): ضریب منفی نشان‌دهنده کاهش احتمال بقا در کلاس ۰ برای جنسیت زنانه است. نسبت شانس کمتر از ۱ بیانگر این است که مردان احتمال بیشتری برای بقا در این کلاس دارند.
- Age (0.0815, Odds Ratio: 1.0849): ضریب مثبت نشان‌دهنده این است که با افزایش سن، احتمال بقا در کلاس ۰ افزایش می‌یابد. نسبت شانس بیشتر از ۱ نشان می‌دهد که افراد مسن‌تر احتمال بیشتری برای بقا در این کلاس دارند.
- SibSp (-1.0569, Odds Ratio: 0.3475): ضریب منفی نشان‌دهنده کاهش احتمال بقا در کلاس ۰ با افزایش تعداد خواهر و برادر یا همسران است. نسبت شانس کمتر از ۱ بیانگر این است که داشتن همراهان بیشتر، احتمال بقا در کلاس ۰ را کاهش می‌دهد.

- Fare (-6.7133, Odds Ratio: 0.0012): ضریب منفی و نسبت شانس بسیار کمتر از ۱ نشان‌دهنده کاهش شدید احتمال بقا در این کلاس با افزایش Fare (قیمت بلیط) است. این نشان می‌دهد که افراد با بلیط‌های گران‌تر بیشتر در کلاس‌های بالاتر (یعنی کلاس‌های با شانس بقا بیشتر) قرار می‌گیرند.
- Embarked_S (-0.1060, Odds Ratio: 0.8995): ضریب منفی نشان‌دهنده کاهش احتمال بقا در این کلاس برای افرادی است که از بندر S سوار شده‌اند. این اثر چندان بزرگ نیست، اما نشان‌دهنده تأثیر جزئی بندر مبدا بر احتمال بقا در کلاس ۰ است.

کلاس ۱(Medium)

- Pclass (-0.1700, Odds Ratio: 0.8436): ضریب منفی نشان‌دهنده کاهش احتمال بقا در کلاس ۱ (شانس بقا متوسط) با افزایش کلاس بلیط است. به عبارت دیگر، افرادی که بلیط‌های ارزان‌تر دارند احتمال بیشتری دارند که در این کلاس قرار گیرند.
- Sex (0.4874, Odds Ratio: 1.6281): ضریب مثبت نشان‌دهنده این است که زنان احتمال بیشتری برای بقا در کلاس ۱ دارند. نسبت شانس بیشتر از ۱ به این معناست که جنسیت زنانه احتمال بقا را در این کلاس افزایش می‌دهد.
- Age (-0.0881, Odds Ratio: 0.9157): ضریب منفی نشان‌دهنده کاهش احتمال بقا در این کلاس با افزایش سن است. افراد مسن‌تر احتمال کمتری دارند که در این کلاس قرار گیرند.
- SibSp (0.0510, Odds Ratio: 1.0524): ضریب مثبت نشان‌دهنده این است که افرادی که تعداد بیشتری خواهر و برادر یا همسر دارند، احتمال بیشتری برای بقا در کلاس ۱ دارند. این می‌تواند به معنی این باشد که افرادی که همراهان بیشتری دارند، در این کلاس بیشتر بقا می‌یابند.
- Fare (0.2663, Odds Ratio: 1.3051): ضریب مثبت نشان‌دهنده این است که افرادی که بلیط گران‌تری دارند احتمال بیشتری برای بقا در کلاس ۱ دارند. نسبت شانس بیشتر از ۱ نشان‌دهنده این است که Fare به افزایش احتمال بقا در این کلاس کمک می‌کند.
- Embarked_S (0.3337, Odds Ratio: 1.3962): ضریب مثبت نشان‌دهنده این است که افرادی که از بندر S سوار شده‌اند، احتمال بیشتری برای بقا در کلاس ۱ دارند. این نشان‌دهنده تأثیر بندر مبدا بر احتمال بقا در این کلاس است.

کلاس ۲(High)

- Pclass (-1.9089, Odds Ratio: 0.1482): ضریب منفی و نسبت شانس کمتر از ۱ نشان‌دهنده کاهش شدید احتمال بقا در کلاس ۲ (شانس بقا زیاد) با افزایش کلاس بلیط است. افراد با بلیط‌های ارزان‌تر بیشتر احتمال دارند که در این کلاس قرار گیرند.

- Sex (0.0435, Odds Ratio: 1.0444): ضریب مثبت نشان‌دهنده این است که زنان احتمال بیشتری برای بقا در این کلاس دارند. با این حال، چون نسبت شانس بسیار نزدیک به ۱ است، تأثیر این ویژگی بسیار جزئی است.
- Age (0.0066, Odds Ratio: 1.0066): ضریب مثبت نشان‌دهنده این است که با افزایش سن، احتمال بقا در این کلاس به طور جزئی افزایش می‌یابد. این اثر بسیار کم است و نسبت شانس نزدیک به ۱ است.
- SibSp (1.0059, Odds Ratio: 2.7343): ضریب مثبت و نسبت شانس بسیار بالا نشان‌دهنده تأثیر بسیار زیاد تعداد خواهر و برادر یا همسران بر احتمال بقا در کلاس ۲ است. این نشان‌دهنده اهمیت همراهان خانوادگی در این کلاس است.
- Fare (6.4470, Odds Ratio: 630.8118): ضریب مثبت و نسبت شانس بسیار بالا نشان‌دهنده تأثیر قوی Fare بر افزایش احتمال بقا در کلاس ۲ است. افرادی که بلیط‌های گران‌تری دارند احتمال بسیار بیشتری برای بقا در این کلاس دارند.
- Embarked_S (-0.2278, Odds Ratio: 0.7963): ضریب منفی نشان‌دهنده کاهش احتمال بقا در این کلاس برای افرادی است که از بندر S سوار شده‌اند. نسبت شانس کمتر از ۱ بیانگر این است که افرادی که از این بندر سوار شده‌اند احتمال کمتری برای بقا در این کلاس دارند.

نتیجه‌گیری کلی

- برای کلاس ۰(Low)، ویژگی‌هایی مانند Fare و SibSp تأثیر منفی دارند و احتمال بقا را کاهش می‌دهند، در حالی که ویژگی‌هایی مانند Pclass و Age تأثیر مثبت دارند و احتمال بقا را افزایش می‌دهند.
- برای کلاس ۱(Medium)، ویژگی‌هایی مانند Sex و Embarked_S تأثیر مثبت دارند و Age تأثیر منفی دارد.
- برای کلاس ۲(High)، ویژگی‌هایی مانند Fare و SibSp تأثیر بسیار مثبت دارند و احتمال بقا را به میزان زیادی افزایش می‌دهند.

این تحلیل نشان می‌دهد که فاکتورهایی همچون قیمت بلیط و همراهان خانوادگی تأثیر زیادی بر احتمال بقا در هر یک از کلاس‌ها دارند. همچنین، ویژگی‌هایی مانند Pclass و Sex نقش مهمی در تعیین شانس بقا دارند.

سوال۲)

۱,۲) همانند سوال یک، فایل داده‌ها را در گوگل کولب آپلود می‌کنیم.

۱,۱,۲) داده ها را به یک دیتافرم تبدیل کرده و در خروجی با متد head(). چاپ می‌کنیم.

```
# Read the CSV file and convert it into a DataFrame
df = pd.read_csv("/content/mp1_lr_dataset_ai4032.csv")

# Print the first 5 rows of the DataFrame in the terminal
print(df.head())
```

```
      1     2     3     4     5     6     7     8     9    10  ...   1991  \
0    13    29    44    55    61    72   200    96   112   120  ...  21680
1 -3980 -3883 -3832     0 -3839 -3788 -3695 -3663 -3669     0  ...   7130
2 -3959 -4058 -4131 -4033 -4009 -3941 -3996 -4071 -3971 -4021  ...   -254

      1992   1993   1994   1995   1996   1997   1998   1999   2000
0  21692  21697  21705  21711  21729  21742  21746  21763  21777
1   7150   7117   7023   7064      0   7027   6929   6954   6867
2   -296   -303      0   -327   -274   -358   -278   -346   -299

[3 rows x 2000 columns]
```

۲,۱,۲) تبدیل داده‌ها به دیتافرم باعث می‌شود:

داده‌ها ساختار جدولی پیدا کنند که هر سطر یک نمونه داده و هر ستون نشان دهنده یک ویژگی است.

ستون ها نام گذاری می‌شوند و می‌توان با نام هرکدام به ستون مورد نظر دسترسی پیدا کرد.

امکان انجام عملیاتی مانند انتخاب داده‌های خاص، خلاصه‌سازی داده‌ها، مرتب کردن بر اساس یک ستون، جایگزینی، تبدیل نوع داده و تغییر فرمت، روی داده‌ها فراهم می‌شود.

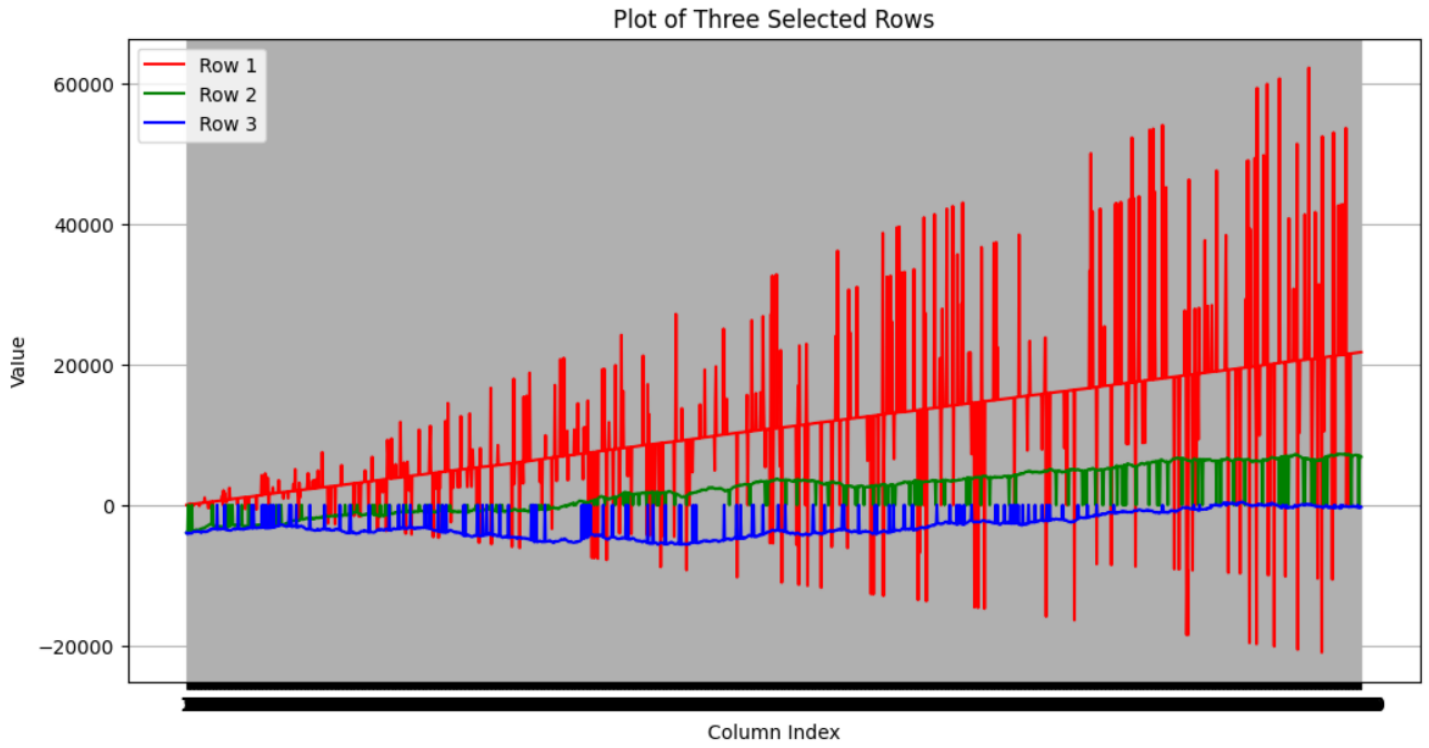
۳,۱,۲) در این بخش داده‌ها را ستونی کرده و آرایه بدست آمده را به یک دیتافرم تبدیل می‌کنیم.

```
# Convert data to a column array
column_array = df.values.reshape(-1, 1)

# Print the column array
#print(column_array)

# Convert the array to a DataFrame
df = pd.DataFrame(column_array, columns=['Column'])

# Display the DataFrame
print(df)
```

```
      Column
0       13.0
1       29.0
2       44.0
3       55.0
4       61.0
...      ...
5995  -274.0
5996  -358.0
5997  -278.0
5998  -346.0
5999  -299.0

[6000 rows x 1 columns]
```

۴،۱،۲) در این بخش با کمک iloc() به ردیف های مورد نظر دسترسی پیدا می‌کنیم و آنها را جداگانه در سه متغیر می‌گذاریم. به هر ردیف یک رنگ و برچسب اختصاص می‌دهیم و آنها را رسم می‌کنیم. با کمک plt.legend() راهنمای نمودار را مشخص می‌کنیم:

```
df = pd.read_csv('/content/mp1_lr_dataset_ai4032.csv')

# Select three different rows (e.g., rows 1, 2, and 3)
row1 = df.iloc[0]
row2 = df.iloc[1]
row3 = df.iloc[2]

# Create the plot
plt.figure(figsize=(12, 6))

# Plot each row with a different color
plt.plot(row1, color='red', label='Row 1')
plt.plot(row2, color='green', label='Row 2')
plt.plot(row3, color='blue', label='Row 3')

# Add a title and axis labels
plt.title('Plot of Three Selected Rows')
plt.xlabel('Column Index')
plt.ylabel('Value')

# Display the legend
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```



۵,۱,۲) تعداد سطرها که نشان دهنده نمونه‌هاست به نسبت تعداد ویژگی‌ها (تعداد ستون‌ها) بسیار کم است و باعث می‌شود به راحتی نتوان الگویی میان آنها پیدا کرد.

وجود داده‌های صفر و داده‌هایی با مقیاس های بسیار بزرگ یا کوچک که باید استاندارد و نرمالیز بشوند.

۲,۲)

۱,۲,۲) داده‌های خام معمولاً دارای مشکلاتی مانند مقادیر گمشده، نویز، داده‌های پرت، مقادیر تکراری یا اشتباهات ثبت داده هستند. اگر این مشکلات برطرف نشوند، می‌توانند باعث عملکرد ضعیف مدل‌های یادگیری ماشین شوند.

اهمیت پاکسازی داده‌ها:

۱. **افزایش دقت مدل‌ها**: مدل‌های یادگیری ماشین به داده‌های باکیفیت نیاز دارند. داده‌های نامناسب باعث یادگیری الگوهای نادرست و پیش‌بینی‌های اشتباه می‌شوند.
۲. **کاهش نویز و خطاها**: داده‌های دارای خطا (مانند ورودی‌های نادرست یا داده‌های پرت) می‌توانند مدل را گمراه کنند. حذف این خطاها باعث افزایش قابلیت اطمینان مدل می‌شود.
۳. **بهبود کارایی پردازش و سرعت مدل**: حذف داده‌های غیرضروری یا اصلاح داده‌های نامعتبر، پردازش داده‌ها را بهینه می‌کند. مدل‌ها سریع‌تر اجرا شده و منابع محاسباتی کمتری مصرف می‌کنند.

۴. **افزایش قابلیت تعمیم مدل**: اگر داده‌ها تمیز نباشند، مدل به جای یادگیری الگوهای عمومی، داده‌های نویزی را حفظ می‌کند.(Overfitting) . با پاکسازی داده‌ها، مدل بهتری برای پیش‌بینی روی داده‌های جدید ساخته می‌شود.

در این بخش ابتدا دو عدد را برای آستانه پایین و بالای داده‌ها پیدا می‌کنیم و سپس با کمک آنها داده‌ها را به دو روش پاکسازی می‌کنیم:

از کتابخانه‌ی scipy، ماژول statsرو وارد می‌کنیم که ابزارهای آماری مثل Z-score داره.

در DATA = 0 شماره ردیف موردنظر برای تحلیل مشخص می‌کنیم. اینجا ردیف ۰ انتخاب شده.

آستانه‌ای برای Z-score تنظیم می‌کنیم. مقادیری که Z اون‌ها بزرگ‌تر از ۳ یا کمتر از -۳ باشند، پرت حساب می‌شوند.

در row_data = df.iloc[DATA].values، داده‌های ردیف مورد نظر رو به صورت آرایه عددی (numpy array) می‌گیریم.

در z_scores = stats.zscore(row_data)، این قسمت شاخصی برای فاصله عدد از میانگین است که باتوجه به انحراف معیار تعیین می‌شود.

در ادامه میانگین و انحراف معیار ردیف را حساب می‌کنیم تا بعداً آستانه‌های مورد نظر را بسازیم.

در lower_threshold = mean - z_threshold * std

upper_threshold = mean + z_threshold * std آستانه‌ها محاسبه می‌شوند و در ادامه داده‌ها را پاکسازی می‌کینم.

در df_z_cleaned = row_data[np.abs(z_scores) < z_threshold] داده‌هایی که قدر مطلقZ در آن‌ها کمتر از آستانه هست، نگه داشته می‌شوند و بقیه حذف می‌شوند.

```
from scipy import stats

# فرض: داده‌ها قبلاً لود شدن و df موجوده
# مثلا:
# df = pd.read_csv("/content/mp1_lr_dataset_ai4032.csv")

DATA = 0  # ردیف مورد نظر برای تحلیل (مثلاً ردیف اول)
z_threshold = 3  # آستانه Z برای حذف مقادیر پرت

# مرحله ۱: محاسبه Z-score برای ردیف انتخابشده
row_data = df.iloc[DATA].values
z_scores = stats.zscore(row_data)

# مرحله ۲: محاسبه آستانه‌های عددی واقعی
mean = np.mean(row_data)
std = np.std(row_data)

lower_threshold = mean - z_threshold * std
```

```
upper_threshold = mean + z_threshold * std

# مرحله ۳: اعمال پاکسازی با Z-score
df_z_cleaned = row_data[np.abs(z_scores) < z_threshold]

# مرحله ٤: چاپ آستانه‌ها و خلاصه
print(f"📉 Lower numeric threshold (Z<{z_threshold}): {lower_threshold}")
print(f"📈 Upper numeric threshold (Z<{z_threshold}): {upper_threshold}")
print(f"✅ Cleaned data length: {len(df_z_cleaned)} / Original: {len(row_data)}")
```

```
📉 Lower numeric threshold (Z<3): -17643.491233311834
📈 Upper numeric threshold (Z<3): 39658.35023331183
✅ Cleaned data length: 1953 / Original: 2000
```

بعد از یافتن آستانه بالا و پایین، تابعی می‌نویسیم تا داده‌های پرت را پیدا کرده و حذف کند.

cleaned_rows = [] یک لیست خالی تولید می‌کند تا ردیف‌های پاکسازی شده در آن ذخیره شود.

یک حلقه می‌نویسیم تا با کمک data.iterrows() سطرهای داده را بررسی کند(_به جای اندیس سطرها قرار گرفته است، زیرا نیازی به استفاده از اندیس نداریم.).

در ادامه، سطر پاکسازی شده را طوری تعریف می‌کنیم که تنها داده‌هایی که از آستانه بالا، کوچکتر و از آستانه پایین، بزرگتر هستند را نگه می‌دارد.

سطر پاکسازی شده را به لیست خالی اضافه می‌کنیم. با کمک pd.concat داده‌های پاکسازی شده را به یکدیگر وصل کرده و ترانهاده آن را محاسبه می‌کنیم که فرم داده‌ها سطری بشود زیرا pd.concat داده‌ها را به صورت ستونی بهم وصل می‌کند.

داده‌های اصلی را به تابع می‌دهیم تا پاکسازی شوند و داده‌های پرت حذف شود و بعد از حذف آن ها را نمایش می‌دهیم:

```
# Define a function to remove outliers from rows in the DataFrame
def remove_outliers_from_rows(data, lower_threshold=-17643.49,
upper_threshold=39658.35):
    cleaned_rows = []  # Initialize a list to store rows without outliers

    # Iterate through each row in the DataFrame
    for _, row in data.iterrows():
        # Filter values within the specified thresholds
        cleaned_row = row[(row >= lower_threshold) & (row <= upper_threshold)]
        cleaned_rows.append(cleaned_row)  # Add the cleaned row to the list

    # Combine all cleaned rows into a new DataFrame and transpose the result
    cleaned_data = pd.concat(cleaned_rows, axis=1).transpose()
```

```
    return cleaned_data  # Return the cleaned DataFrame

# Read the CSV file (assuming it has a header row)
#data = pd.read_csv('/content/mp1_lr_dataset_ai4032.csv')

# Apply the function to remove outliers from the data
cleaned_data_nan = remove_outliers_from_rows(df)

# Print the resulting cleaned data
print(cleaned_data_nan)
```

```
        1       2       3       4       5       6       7       8       9  \
0    13.0    29.0    44.0    55.0    61.0    72.0   200.0    96.0   112.0
1 -3980.0 -3883.0 -3832.0     0.0 -3839.0 -3788.0 -3695.0 -3663.0 -3669.0
2 -3959.0 -4058.0 -4131.0 -4033.0 -4009.0 -3941.0 -3996.0 -4071.0 -3971.0

       10  ...    1991    1992    1993    1994    1995   1996    1997    1998  \
0   120.0  ...     NaN     NaN     NaN     NaN     NaN    NaN     NaN     NaN
1     0.0  ...  7130.0  7150.0  7117.0  7023.0  7064.0    0.0  7027.0  6929.0
2 -4021.0  ...  -254.0  -296.0  -303.0     0.0  -327.0 -274.0  -358.0  -278.0

     1999    2000
0     NaN     NaN
1  6954.0  6867.0
2  -346.0  -299.0

[3 rows x 2000 columns]
```

در ادامه روش دوم پاکسازی که جایگزینی داده‌های پرت با میانگین داده‌های کناریست را پیش می‌گیریم:

ابتدا یک تابع تعریف می‌کنیم که پاکسازی را برای ما انجام دهد:

از data یک کپی به نام cleaned_data می‌گیریم تا تغییرات روی نسخه‌ی کپی اعمال شود و داده‌های اصلی تغییری نکند.

یک حلقه می‌نویسیم تا ستون‌های دیتافرم را برای پیدا کردن داده پرت بررسی کند.

داده‌ها را به اعداد اعشاری تبدیل می‌کنیم تا میانگین به درستی محاسبه شود.

با یک حلقه ستون‌های داده‌ها را به نوبت در نظر می‌گیریم، شرط اینکه مقدار داده در بازه مورد قبول(بین دو آستانه) باشد را بررسی کرده، یک لیست خالی با نام همسایه تعریف می‌کنیم و سپس بررسی می‌کنیم که اگر ستون در حال بررسی، ستون اول یا آخر نباشد، همسایه‌های آن را درون لیست خالی ذخیره می‌کنیم.

میانگین همسایه‌ها را در صورت وجود محاسبه کرده و با داده پرت جایگزین می‌کنیم. و اگر همسایه‌ای وجود نداشته باشد، میانگین ستون مربوطه را به جای داده قرار می‌دهیم و اگر میانگین خارج از بازه مورد نظر باشد، همان مقدار آستانه بالا جایگزین داده می‌شود.

در نهایت تابع، داده پاکسازی شده را برمی‌گرداند.

سپس داده‌هایی که میخواستیم پاکسازی کنیم را به تابع نوشته شده می‌دهیم:

```
import pandas as pd

def replace_outliers_with_neighbors_mean_rows(data, lower_threshold=-17643.49,
upper_threshold=39658.35):

    cleaned_data = data.copy()  # Create a copy of the input data to modify
    for col in cleaned_data.columns:  # Process each column
        cleaned_data[col] = cleaned_data[col].astype(float)  # Convert data to float
type
        for j in range(len(cleaned_data)):  # Process each row
            if cleaned_data.loc[j, col] < lower_threshold or cleaned_data.loc[j, col]
> upper_threshold:
                # Identify outlier value
                neighbors = []
                if j > 0:
                    neighbors.append(cleaned_data.loc[j - 1, col])  # Previous value
                if j < len(cleaned_data) - 1:
                    neighbors.append(cleaned_data.loc[j + 1, col])  # Next value

                if neighbors:  # If there are neighbors
                    mean_neighbors = round(sum(neighbors) / len(neighbors))  #
Calculate mean of neighbors
                else:
                    mean_neighbors = cleaned_data[col].mean()  # Otherwise, use the
column mean

                # Ensure the value stays within the threshold limits
                cleaned_data.loc[j, col] = max(lower_threshold, min(mean_neighbors,
upper_threshold))

    return cleaned_data  # Return the cleaned dataset

# Read data from CSV file
df = pd.read_csv('/content/mp1_lr_dataset_ai4032.csv')

# Apply the function to clean data
cleaned_data_mean = replace_outliers_with_neighbors_mean_rows(df)

# Get the maximum value of the cleaned dataset
#overall_max = cleaned_data_mean.values.max()
#print("\n📌 Maximum value of the cleaned dataset:", overall_max)


#print(cleaned_data_mean)
```

```
print(cleaned_data_mean)
```

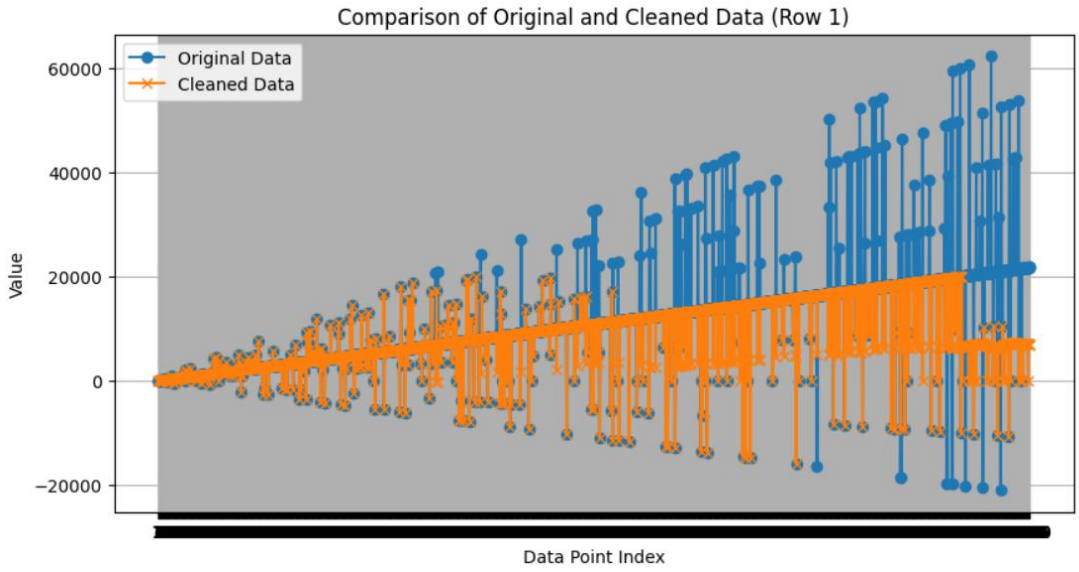
```
        1       2       3       4       5       6       7       8       9  \
0    13.0    29.0    44.0    55.0    61.0    72.0   200.0    96.0   112.0
1 -3980.0 -3883.0 -3832.0     0.0 -3839.0 -3788.0 -3695.0 -3663.0 -3669.0
2 -3959.0 -4058.0 -4131.0 -4033.0 -4009.0 -3941.0 -3996.0 -4071.0 -3971.0

       10  ...    1991    1992    1993    1994    1995   1996    1997    1998  \
0   120.0  ...  7130.0  7150.0  7117.0  7023.0  7064.0    0.0  7027.0  6929.0
1     0.0  ...  7130.0  7150.0  7117.0  7023.0  7064.0    0.0  7027.0  6929.0
2 -4021.0  ...  -254.0  -296.0  -303.0     0.0  -327.0 -274.0  -358.0  -278.0

     1999    2000
0  6954.0  6867.0
1  6954.0  6867.0
2  -346.0  -299.0

[3 rows x 2000 columns]
```

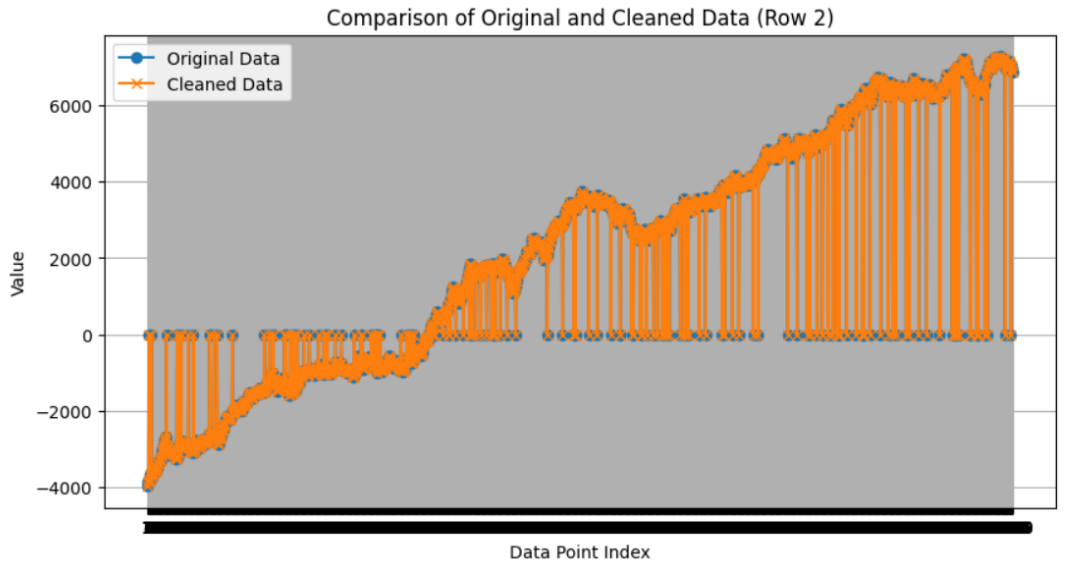
در این بخش، داده‌ها را به روش جایگزینی با میانگین پاکسازی کرده و هر سطر از داده اصلی را با هر سطر از داده پاکسازی شده رسم کرده و مقایسه می‌کنیم:

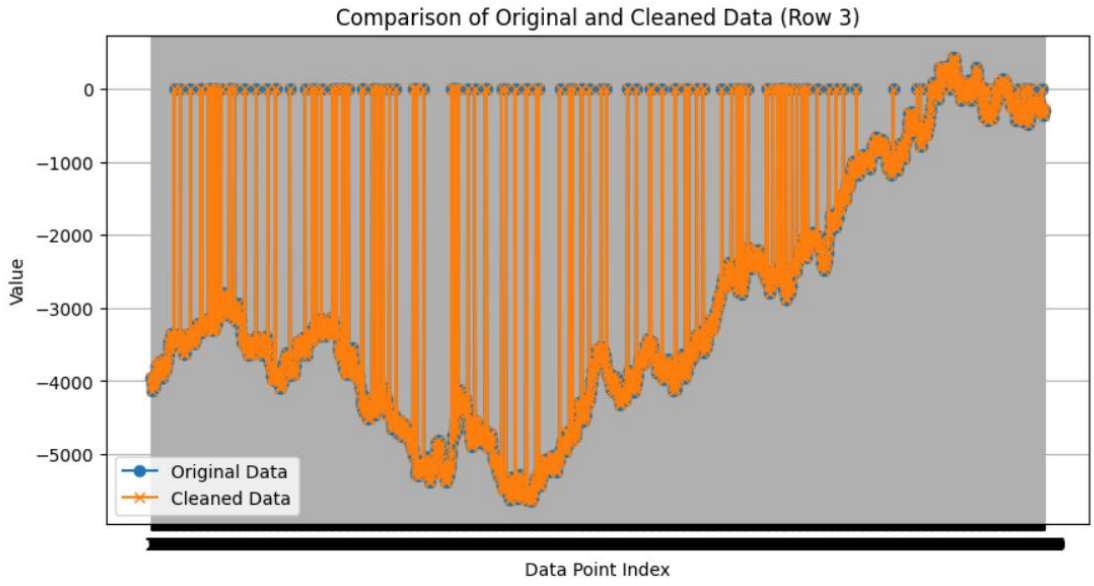
```
#df = pd.read_csv('/content/mp1_lr_dataset_ai4032.csv')

# Clean the data
#cleaned_data = replace_outliers_with_neighbors_mean_rows(df)

# Plot for comparison
plt.figure(figsize=(10, 5))

# Plot original data
plt.plot(df.iloc[0], label='Original Data', marker='o')

# Plot cleaned data
plt.plot(cleaned_data_mean.iloc[0], label='Cleaned Data', marker='x')

plt.title('Comparison of Original and Cleaned Data (Row 1)')
plt.xlabel('Data Point Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
#####################################################################
# Plot for comparison
plt.figure(figsize=(10, 5))

# Plot original data
```

```
plt.plot(df.iloc[1], label='Original Data', marker='o')

# Plot cleaned data
plt.plot(cleaned_data_mean.iloc[1], label='Cleaned Data', marker='x')

plt.title('Comparison of Original and Cleaned Data (Row 2)')
plt.xlabel('Data Point Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
######################################################################
# Plot for comparison
plt.figure(figsize=(10, 5))

# Plot original data
plt.plot(df.iloc[2], label='Original Data', marker='o')

# Plot cleaned data
plt.plot(cleaned_data_mean.iloc[2], label='Cleaned Data', marker='x')

plt.title('Comparison of Original and Cleaned Data (Row 3)')
plt.xlabel('Data Point Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```

در ادامه با مشاهده خروجی متوجه می‌شویم که تنها ردیف اول دارای داده پرت و خارج از آستانه بوده است:







۲,۳ و ۲,۴)

2.3.1

ساختن رگرسیون خطی از اسکرچ یعنی بدون استفاده از کتابخانه های مربوطه، عمل رگرسیون را انجام دهیم ، و از آنجایی که سوال از ما میخواهد که به دو روش این کار صورت گیرد پس ما از دو روش LS (Least Square) و گرادیان نزولی (Gradient Descent) استفاده میکنیم و این کار را برای هر سه ردیف داده ها باید انجام دهیم

برای پیاده سازی این پروسه برای ردیف اول باید آن ردیف را از داده های مورد نظر انتخاب کنیم که دستور آن به صورت زیر خواهد بود

```
first_row = cleaned_data_mean.iloc[0, :].values
```

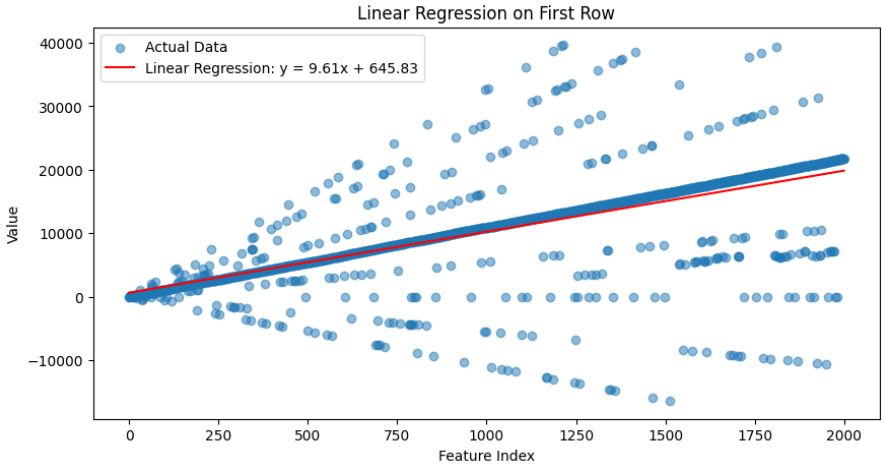
سپس باید داده ها و خروجی ها ما یه ترتیب X و Y را انتخاب کنیم. از آنجایی که میخواهیم داده ها را بررسی کنیم پس محور X را اندیس داده درنظر میگیریم و محور Y را داده های هر اندیس و برای اینکه معادله بین داده ها دارای عرض از مبدا باشند و مبدا گذر نباشند باید به معادله یک بایاس اضافه کنیم تا بتوانیم معادله $(X^T X)^{-1} X^T Y$ را پیاده کنیم. حال مدل ما تحت الگوریتم بالا آموزش داده میشود و میتواند مقادیر خروجی ها را پیش بینی کند پس کد ما باید به صورت زیر باشد

```
# Use the cleaned data
first_row = cleaned_data_mean.iloc[0, :].values  # Extract first row as NumPy array

# Generate X values (assuming sequential indices as features)
X = np.arange(len(first_row)).reshape(-1, 1)  # Reshape to column vector
Y = first_row.reshape(-1, 1)  # Reshape target values

# Compute parameters using Normal Equation: θ = (X^T * X)^(-1) * X^T * Y
X_bias = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term (column of ones)
theta = np.linalg.inv(X_bias.T.dot(X_bias)).dot(X_bias.T).dot(Y)  # Compute theta

# Extract intercept and slope
intercept, slope = theta[0, 0], theta[1, 0]

# Predict Y values
Y_pred = X_bias.dot(theta)
```

حال برای نشان دادن و پلات کردن نتیجه کد بالا بر ردیف اول داده ها در ادامه به این صورت عمل میکنیم

```
# Plot the results
plt.figure(figsize=(10, 5))
plt.scatter(X, Y, label="Actual Data", alpha=0.5)
plt.plot(X, Y_pred, color='red', label=f"Linear Regression: y = {slope:.2f}x +
{intercept:.2f}")
plt.xlabel("Feature Index")
plt.ylabel("Value")
plt.legend()
plt.title("Linear Regression on First Row")
plt.show()

print(f"✅ Model trained successfully: y = {slope:.4f}x + {intercept:.4f}")
```

نتیجه کد:



حال میخواهیم همین کد را برای ردیف دوم و سوم نیز استفاده کنیم برای اینکار کافی است فقط مقادیر کد را از ردیف اول به ردیف دوم و یا سوم تغییر دهیم یعنی کد به صورت زیر تغییر میکند

کد تعیین کننده برای ردیف اول:

```
first_row = cleaned_data_mean.iloc[0, :].values
X = np.arange(len(first_row)).reshape(-1, 1)
Y = first_row.reshape(-1, 1)
```

کد تعیین کننده برای ردیف دوم:

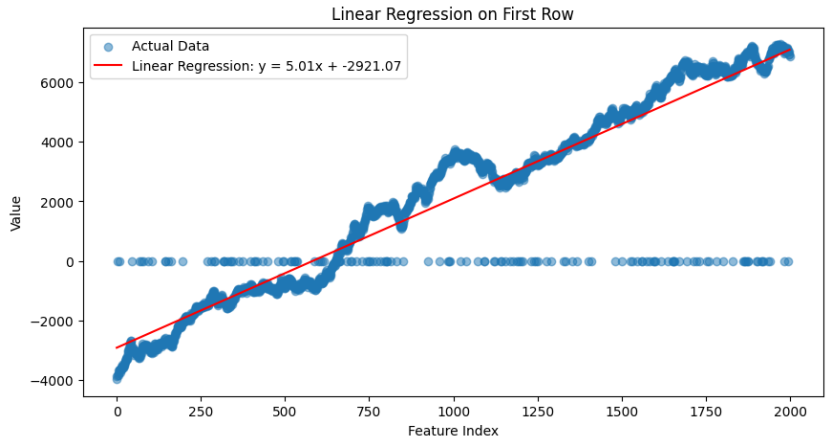
```
second_row = cleaned_data_mean.iloc[1, :].values
X = np.arange(len(second_row)).reshape(-1, 1)
Y = second_row.reshape(-1, 1)
```

کد تعیین کننده برای ردیف سوم:

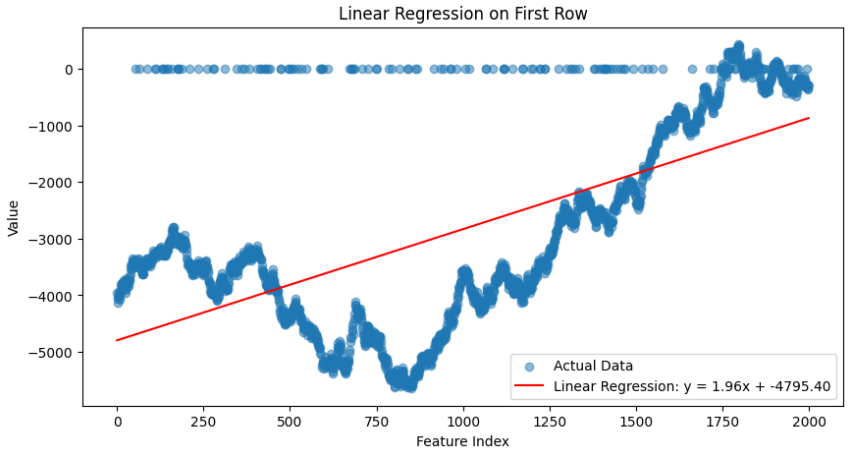
```
third_row = cleaned_data_mean.iloc[2, :].values
X = np.arange(len(third_row)).reshape(-1, 1
Y = third_row.reshape(-1, 1)
```

و نتیجه کد نیز به این صورت خواهد بود

ردیف دوم:



ردیف سوم:



همانطور که میبینیم در ردیف های اول و دوم پراکندگی و تجمع داده ها به صورتی است که میتوان آن را با یک رگرسیون تک متغیره نمایش داد ولی برای ردیف سوم به دلیل نوسانات زیاد در داده ها با وجود اینکه خط رگرسیون کشیده شده است اما خیلی تقریب خوبی به ما نمی دهد

حال میخواهیم دوباره عمل رگرسیون را از اسکرچ انجام دهیم اما با روشی گرادیان نزولی.

ایده اصلی این روش استفاده از تابع هزینه MSE یا میانگین مربعات خطا است و بار هر بار اجرا از خطای کمتر میشود

برای پیاده سازی این روش ابتدا همانند روش LS ردیف اول و X و Y را مشخص میکنیم سپس برای اینکه داده ها را راحت تر تحلیل کنیم از روش نرمال سازی داده ها استفاده میکنیم. در این روش داده نرمال شده حاصل تقسیم اختلاف داده با میانگین کل داده ها بر انحراف معیار است که این باعث می شود الگوریتم پایدارتر شود

سپس همانند بخش قبل به داده ها عرض از مبدا داده که اینکار ستونی از ۱ ها با داده ها اضافه میکند

و به مدل وزن های متفاوت و تصادفی میدهیم که الگوریتم از یک نقطه تصادفی شروع کرده و سریعتر

به جواب برسه

```
theta = np.random.randn(2, 1)
```

چون این الگوریتم با تکرار به جواب میرسد پس باید نرخ یادگیری و تعداد تکرار و نمونه برای آن تعریف گردد

```
learning_rate = 0.01
n_iterations = 1000
m = len(X_bias)
```

حال برای ایجاد حلقه تکرار به صورت زیر عمل میکنیم

```
for iteration in range(n_iterations):
    gradients = (2/m) * X_bias.T.dot(X_bias.dot(theta) - Y)
    theta = theta - learning_rate * gradients
```

که در این کد برای خطا، مقدار پیش بینی شده منهای مقدار واقعی شده و وزن ها که همان تتا ها هستند آپدیت میشوند یعنی فرمول روبه رو پیاده سازی میشود که آلفا در آن همان ضریب یادگیری است

$$\theta = \theta - \alpha . \nabla J(\theta)$$

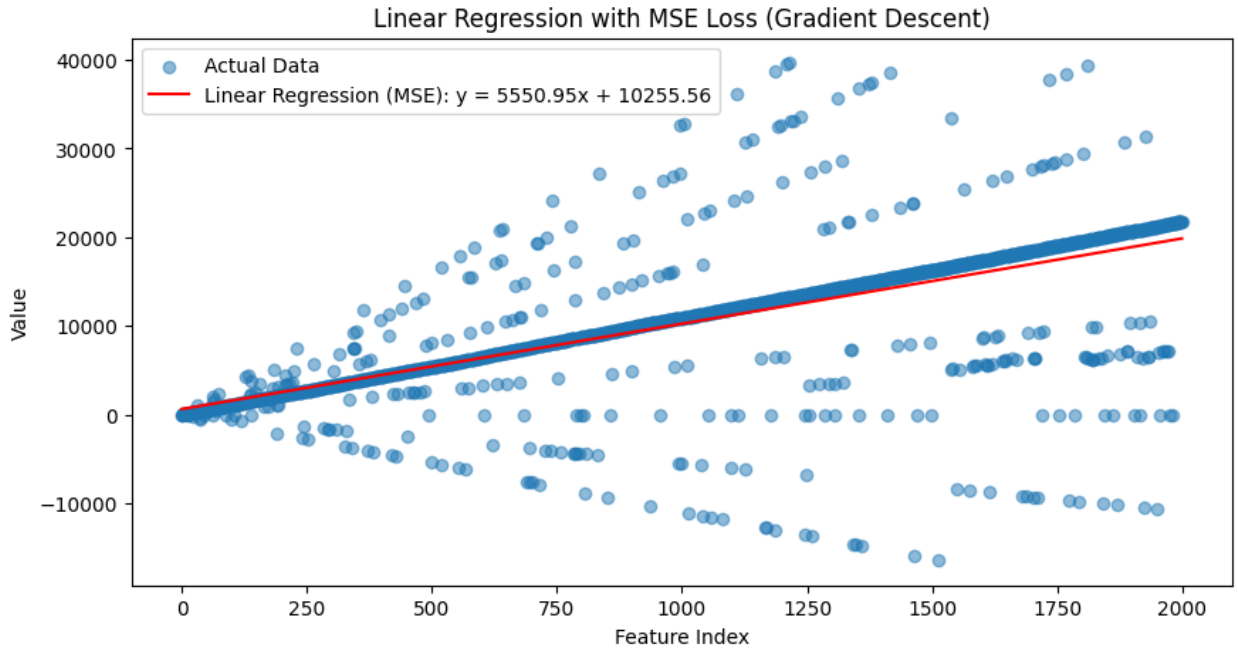
در نهایت نیز به کمک الگوریتم بالا مقادیر جدید را پیش بینی و پلات میکنیم

پس کد به صورت زیر خواهد بود:

```
# Step 1: Prepare data
first_row_GD = cleaned_data_mean.iloc[0, :].values  # First row as NumPy array
X = np.arange(len(first_row_GD)).reshape(-1, 1)
Y = first_row_GD.reshape(-1, 1)
```

```
# Normalize X for better gradient descent stability (optional but helpful)
X_norm = (X - X.mean()) / X.std()

# Add bias term to X
X_bias = np.c_[np.ones((X_norm.shape[0], 1)), X_norm]

# Step 2: Initialize weights
theta = np.random.randn(2, 1)  # [bias, weight]

# Step 3: Set hyperparameters
learning_rate = 0.01
n_iterations = 1000
m = len(X_bias)

# Step 4: Gradient Descent
for iteration in range(n_iterations):
    gradients = (2/m) * X_bias.T.dot(X_bias.dot(theta) - Y)
    theta = theta - learning_rate * gradients

# Extract parameters
intercept, slope = theta[0, 0], theta[1, 0]
Y_pred = X_bias.dot(theta)

# Step 5: Plot results
plt.figure(figsize=(10, 5))
plt.scatter(X, Y, label="Actual Data", alpha=0.5)
plt.plot(X, Y_pred, color='red', label=f"Linear Regression (MSE): y = {slope:.2f}x +
{intercept:.2f}")
plt.xlabel("Feature Index")
plt.ylabel("Value")
plt.legend()
plt.title("Linear Regression with MSE Loss (Gradient Descent)")
plt.show()

# Step 6: Final output
print(f"✅ Model trained with Gradient Descent:")
print(f"Intercept (θ₀) = {intercept:.4f}")
print(f"Slope (θ₁)     = {slope:.4f}")
```

نتیجه کد بالا:



برای پیاده سازی این کد برای ردیف دوم و سوم کافی است همانند بخش قبل پارامتر تعیین کننده سطر اول با سطر دوم و با سوم جایگزین شوند

پارامتر های تعیین کننده سطر اول:

```
first_row_GD = cleaned_data_mean.iloc[0, :].values
X = np.arange(len(first_row_GD)).reshape(-1, 1)
Y = first_row_GD.reshape(-1, 1)
```

پارامتر های تعیین کننده سطر دوم:

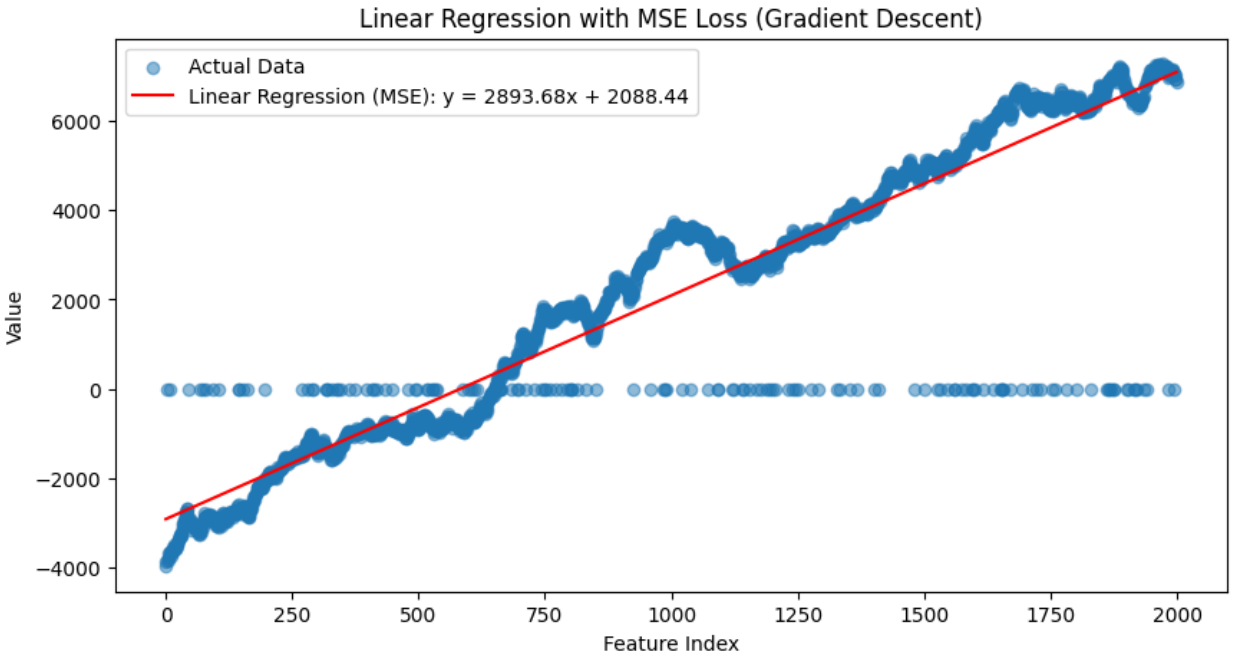
```
second_row_GD = cleaned_data_mean.iloc[1, :].values
X = np.arange(len(second_row_GD)).reshape(-1, 1)
Y = second_row_GD.reshape(-1, 1)
```

پارامتر های تعیین کننده سطر سوم:

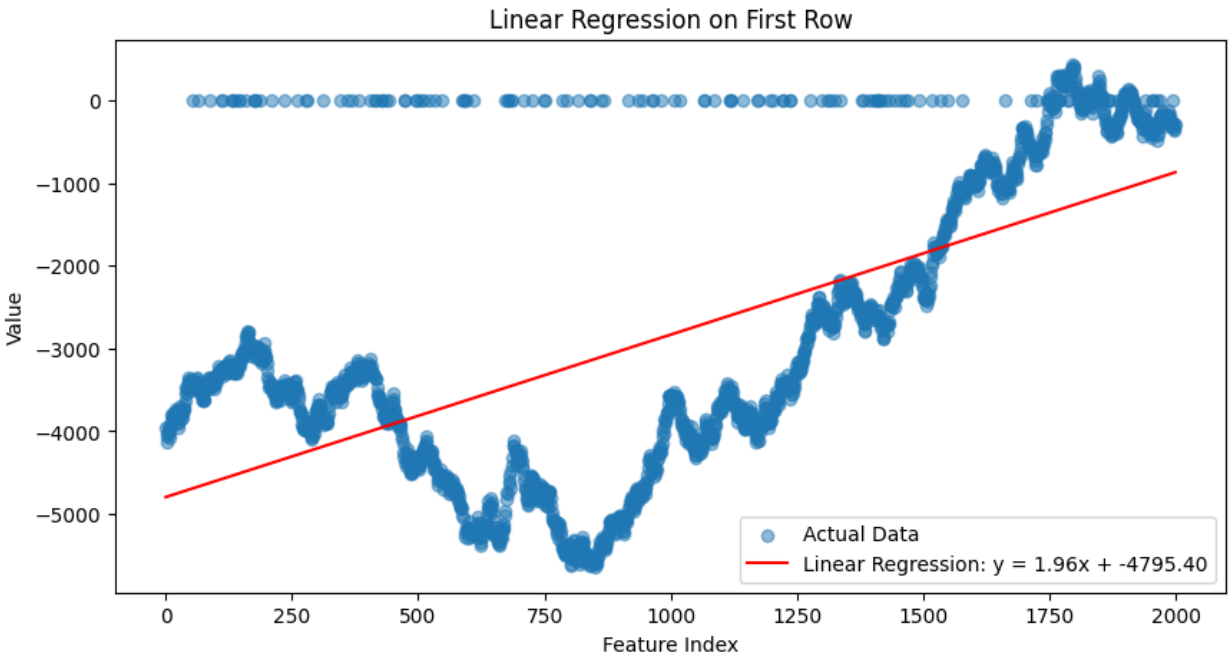
```
Third_row_GD = cleaned_data_mean.iloc[2, :].values
X = np.arange(len(Third_row_GD)).reshape(-1, 1)
Y = Third_row_GD.reshape(-1, 1)
```

پس بنابراین نتیجه کد برای سطر های دوم و سوم یه صورت زیر خواهند بود:

نتیجه کد روش گرادیان نزولی برای ردیف دوم:



نتیجه کد روش گرادیان نزولی برای ردیف سوم:



همانطور که مشاهده میشود نتیجه کد ها به روش گرادیان نزولی شباهت دارند به نتایج در روش LS که نشان از درستی رگرسیون میدهد

تفاوت اصلی بین این دو روش برای رگرسیون این است که در گرادیان نزولی روش با تکرار همراه است و در آن نرمال سازی معمولا صورت میگیرد و برای تحلیل داده های زیادتر مناسب تر است

2.3.2

در این بخش میخواهیم عمل رگرسیون را به کمک کتابخانه sklearn انجام دهیم

تفاوت این روش با روش های قبلی این است که کتابخانه sklearn خیلی سریعتر کار را انجام میده و نیارمند الگوریتم برای انجام رگرسیون و پیداکردن داده ها نیست و میتواند این کار را با دستور fit به راحتی انجام دهد

برای پیاده سازی عمل رگرسیون ابتدا مدل رگرسیون را از sklearn وارد میکنیم و همانند قسمت های قبلی به X مقادیر اندیس ها و به Y مقادیر سطرها را میدهیم و حالا برعکس جالت قبل فقط با دستور linearRegression رگرسیون انجام میشود و با دستور fit نیز مدل با داده ها آموزش داده میشود

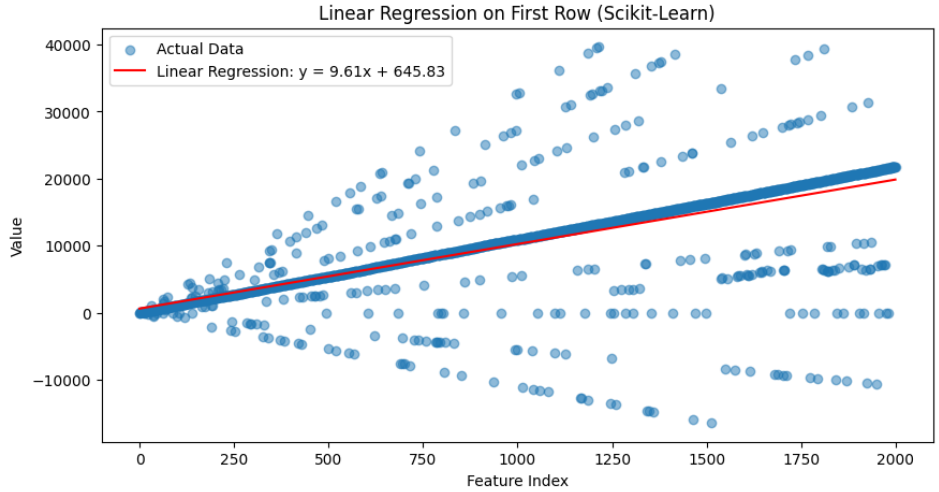
و سپس با مدل آموزش دیده مقادیر را میتوان پیشبینی کرد و در نهایت مدل را رسم میکنیم

پس کد به صورت زیر خواهد بود:

```
from sklearn.linear_model import LinearRegression
# Use the cleaned data
first_row = cleaned_data_mean.iloc[0, :].values  # Extract first row as NumPy array

# Generate X values (column indices as features)
X_1 = np.arange(len(first_row)).reshape(-1, 1)  # Reshape to column vector
Y_1 = first_row.reshape(-1, 1)  # Reshape target values

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_1, Y_1)

# Get predictions
Y_1_pred = model.predict(X_1)

# Extract model parameters
slope = model.coef_[0, 0]
intercept = model.intercept_[0]

# Plot the results
plt.figure(figsize=(10, 5))
plt.scatter(X_1, Y_1, label="Actual Data", alpha=0.5)
plt.plot(X_1, Y_1_pred, color='red', label=f"Linear Regression: y = {slope:.2f}x +
{intercept:.2f}")
plt.xlabel("Feature Index")
plt.ylabel("Value")
plt.legend()
plt.title("Linear Regression on First Row (Scikit-Learn)")
plt.show()
```

```
print(f"✅ Model trained successfully: y = {slope:.4f}x + {intercept:.4f}")
```

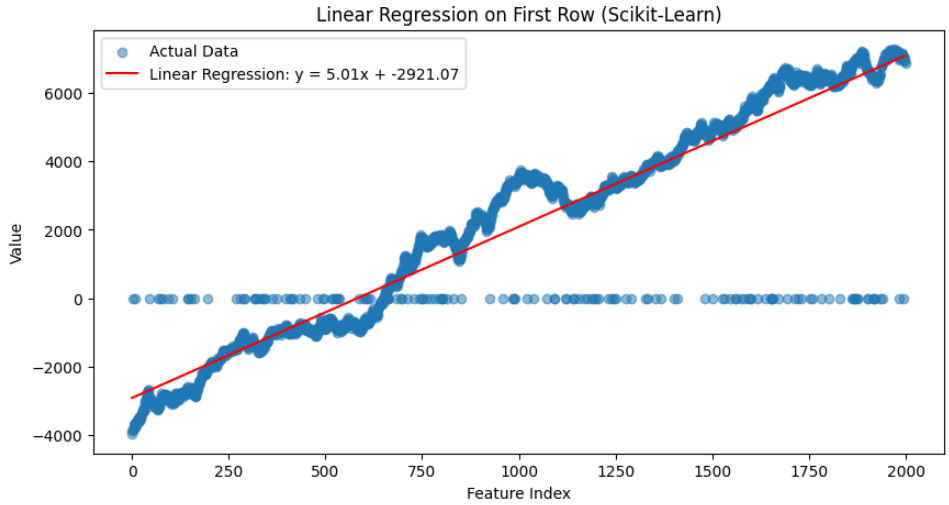
نتیجه کد برای ردیف اول داده ها:



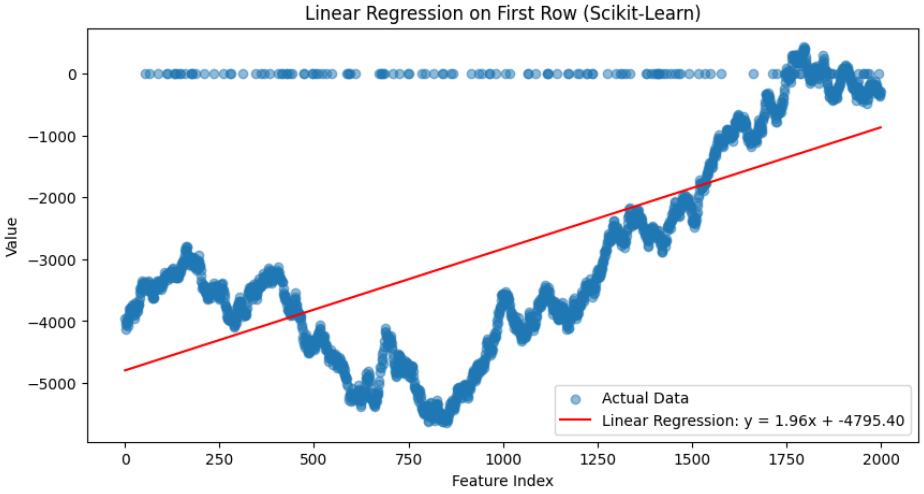
برای بررسی نتیجه ردیف دوم و سوم داده ها نیز مانند قبل تنها کافیست مقدار سطر را در iloc یه ۱ و ۲ تغییر داده و نام متغیر را از first_row به second_row و third_row عوض کنیم

```
second_row = cleaned_data_mean.iloc[1, :].values
third_row = cleaned_data_mean.iloc[2, :]
```

نتیجه کد برای ردیف دوم داده:



نتیجه کد برای ردیف سوم:



با توجه به نتایج بالا و گذشته مدلی که با کتابخانه sklearn آموزش دیده عملکرد بهتری داشته است.

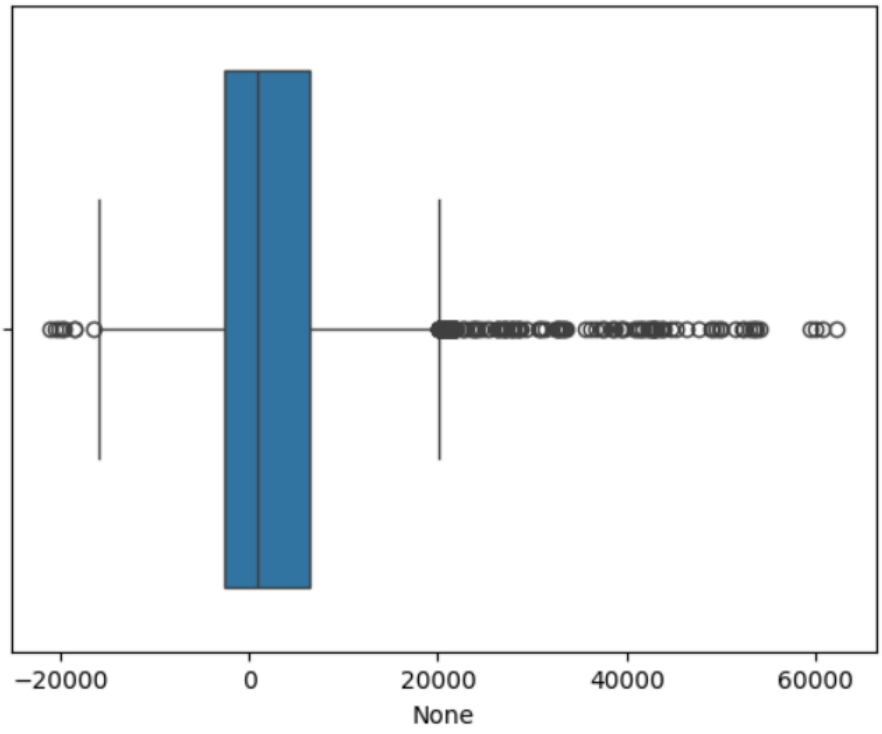
تغییر آستانه

یک تابع تعریف می‌کنیم که آستانه‌های مورد نیاز را پیدا کند. نمودار جعبه‌ای داده‌ها را رسم می‌کنیم تا بتوانیم آستانه را مشاهده کنیم. چارک اول و سوم (به ترتیب یعنی مقدار داده‌ای که ۲۵٪ داده‌ها از آن کوچکتر و مقدار داده‌ای که ۷۵٪ داده‌ها از آن کوچکتر هستند) را محاسبه کرده و از اختلاف آنها ، IQR یعنی محدوده‌ای که ۵۰٪ میانی داده‌ها در آن قرار دارند را پیدا می‌کنیم. با کمک IQR و چارک اول و سوم آستانه پایین و بالای داده‌ها را می‌یابیم.

داده‌ها را به یک آرایه نامپای یک بعدی و سپس به یک سری پانداس تبدیل می‌کنیم. با کمک تابعی که در ابتدا برای محاسبه آستانه نوشتیم، آستانه‌های داده‌ها را پیدا می‌کنیم.

```
def find_outlier_threshold_boxplot(data):
    # Draw a boxplot
    sns.boxplot(x=data)
    plt.show()

    # Calculate quartiles and interquartile range (IQR)
    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    iqr = q3 - q1

    # Calculate thresholds
    lower_threshold = q1 - 1.5 * iqr
    upper_threshold = q3 + 1.5 * iqr
```

```
    return lower_threshold, upper_threshold

# Read the CSV file
#df = pd.read_csv("/content/mp1_lr_dataset_ai4032.csv")

# Convert DataFrame to a one-dimensional NumPy array
column_array = df.values.flatten()

# Convert NumPy array to a pandas series
column_series = pd.Series(column_array)

lower, upper = find_outlier_threshold_boxplot(column_series)
print(f"Lower threshold: {lower}")
print(f"Upper threshold: {upper}")
```



آستانه پایین: -16085.75
آستانه بالا: 20046.25

2.3.3

در رگرسیون معمولی مثل گرادیان نزولی یا Linear regression نوع تابع هزینه میانگین مربعات خطا هستش که یکی از مشکلات این رگرسیون ها حیایست بالای آنها به داده های پرت هستش چون خطاها به توان ۲ میرسند
اما مدل رگرسیون مقاوم ترکیبی از تابع هزینه MAE و MSE هستش که بسته مقدار خطا رفتارش رو عوض میکنه
اگر خطا کوچک باشد مثل MSE یعنی مربعی و اگر خطا بزرگ باشد مثل MAE یعنی خطی رفتار میکنه که این کار تاثیر داده های پرت رو در مدل کمتر میکند

برای انجام رگرسیون مقاوم از دو روش Huber و RANSAC استفاده میشود

در ابتدا به بررسی الگوریتم Huber میپردازیم

در این الگوریتم ابتدا همانند بخش های قبلی ابتدا X وY را مشخص کرده که Y مقادیر سطر اول و X شماره ستون ها یا همان ایندکس ما است سپس مقادیر Nan اگر در داده ها وجود داشته باشد را حذف میکنیم که با خطایی مواجه نشویم

و برای آکوزش مدل فقط کافی است دستور تعریف شده در کتابخانه sklearn را استفاده کنیم تا مدل خواسته شده ایجاد و آموزش صورت گیرد و نتایج را رسم می‌کنیم

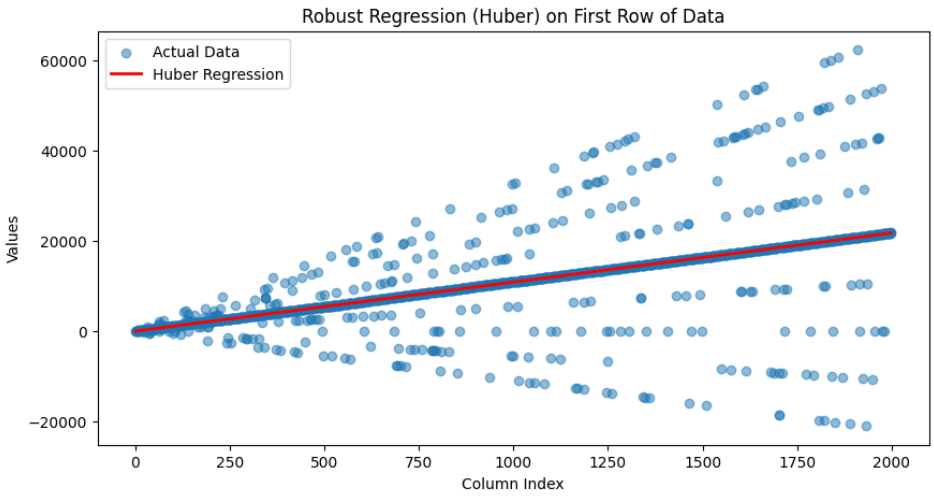
پس کد برای نمایش ردیف اول به صورت زیر خواهد بود

```
from sklearn.linear_model import LinearRegression, HuberRegressor
from sklearn.model_selection import train_test_split
# انتخاب سطر اول
X = np.arange(data.shape[1]).reshape(-1, 1)  # شاخص ستون‌ها به عنوان X
Y = data.iloc[0].values.reshape(-1, 1)  # مقادیر سطر اول به عنوان Y

# حذف مقادیر NaN (در صورت وجود)
mask = ~np.isnan(Y.squeeze())
X, Y = X[mask], Y[mask]

# ایجاد و آموزش مدل HuberRegressor
huber = HuberRegressor()
huber.fit(X, Y.ravel())
Y_pred = huber.predict(X)

# رسم نتایج
plt.figure(figsize=(10, 5))
plt.scatter(X, Y, label="Actual Data", alpha=0.5)
plt.plot(X, Y_pred, color='red', label="Huber Regression", linewidth=2)
```

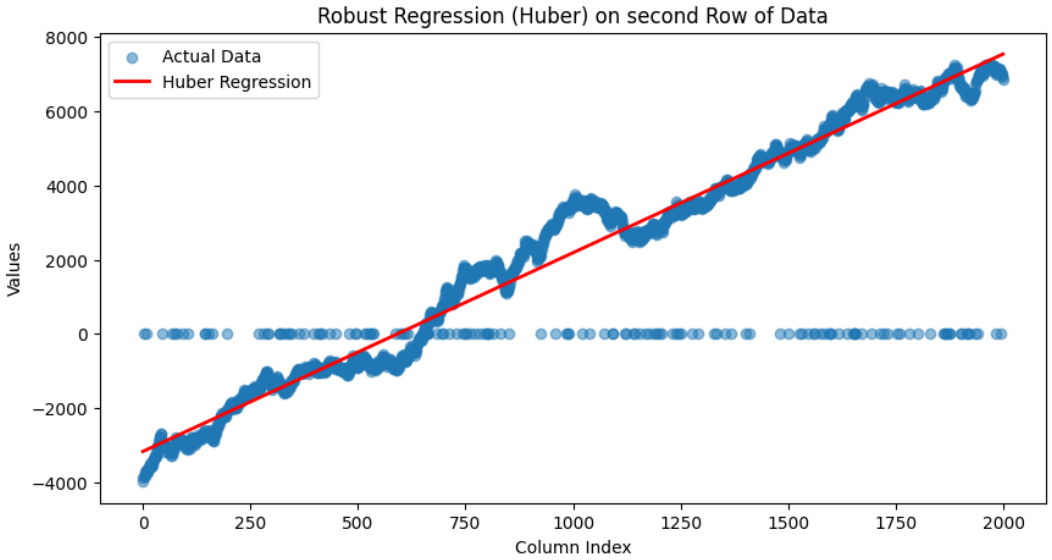
```
plt.xlabel("Column Index")
plt.ylabel("Values")
plt.legend()
plt.title("Robust Regression (Huber) on First Row of Data")
plt.show()
```

و برای دیدن نتیجه برای سطر های دوم و سوم کافی است مقدار عدد دستور iloc را از 0 به 1 برای ردیف دوم و 2 برای ردیف سوم تغییر دهیم، بنابراین نتایج ما به صورت زیر خواهند بود:

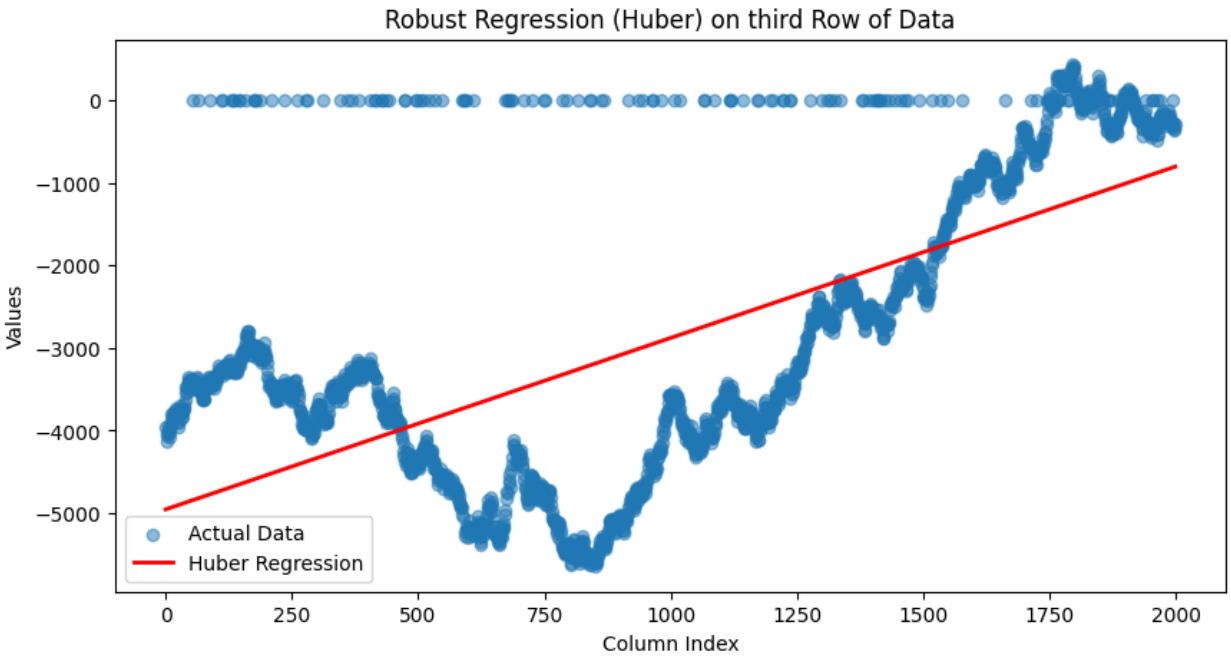
رگرسیون Huber برای ردیف اول:



رگرسیون Huber برای ردیف دوم:



رگرسیون Huber برای ردیف سوم:



حال میخواهیم با نوع دیگری از رگرسیون مقاوم یعنی RANSAC رگرسیون را انجام دهیم

برای پیاده سازی الگوریتم همانند قبلا ابتدا X و Y را مشخص کرده و بعد به تعریف مدل RANSAC میپردازیم

```
ransac = RANSACRegressor(estimator=LinearRegression(), min_samples=50,
residual_threshold=10.0, random_state=42)
ransac.fit(X, Y)
```

در این کد توضیح هر خط به شرح زیر است:

estimator=LinearRegression(): مدل پایه همون خط رگرسیون ساده است.

min_samples=50: حداقل ۵۰ تا نمونه باید برای فیت کردن هر مدل انتخاب بشن.

residual_threshold=10.0: اگه فاصله‌ی نقطه از مدل بیشتر از ۱۰ باشه، اون رو **پرت (Outlier)** در نظر می‌گیرد

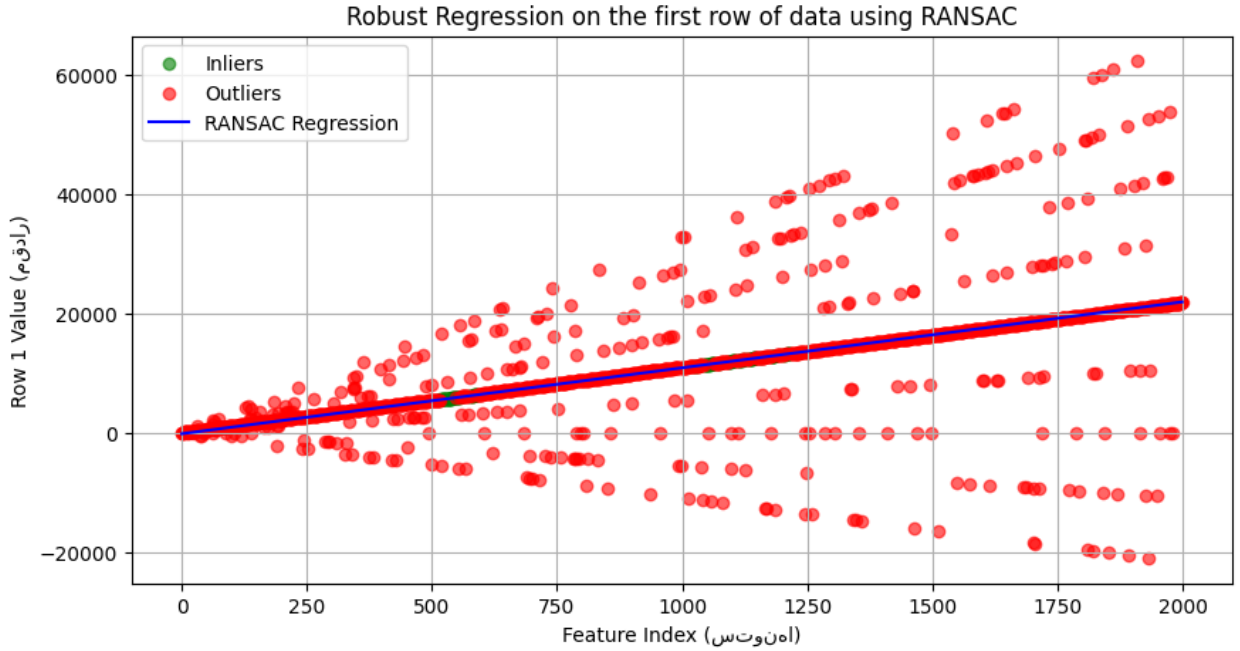
random_state=42): برای تولید تصادفی پایدار

و در نهایت برای آموزش مدل کافی است که دستور ransac.fit(X,Y) را اجرا کنیم که در این دستور زیرمجموعه های کوچک از داده ها انتخاب شده و روی آنها مدل linearRegression فیت میشود و درنهایت مدلی را نگه میدارد که بیشترین تعداد داده دری خط Inliner را تولید کرده باشد و سپس با رنگ سبز اینلاینر های درست را مشخص کرده و پلات میکنیم

پس کد به صورت زیر خواهد بود:

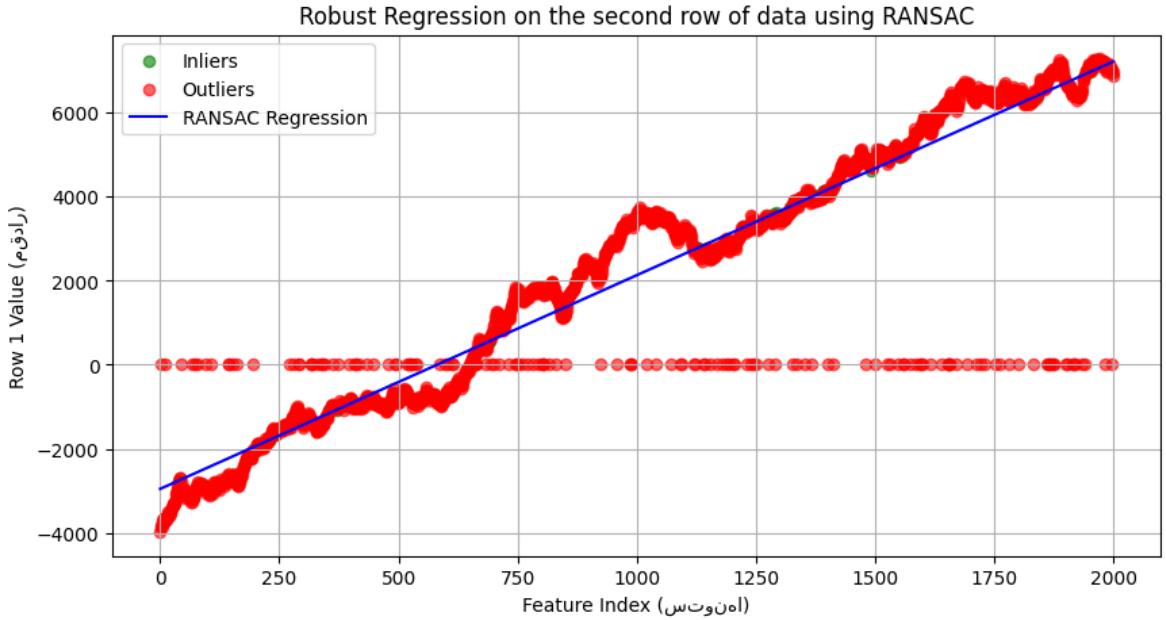
```
#RANSAC method
from sklearn.linear_model import RANSACRegressor, LinearRegression


# داده‌ی سطر اول
Y = data.iloc[0].values.reshape(-1, 1)
X = np.arange(len(Y)).reshape(-1, 1)

# مدل رگرسیون مقاوم با RANSAC
# Instead of base_estimator, use estimator
ransac = RANSACRegressor(estimator=LinearRegression(), min_samples=50,
residual_threshold=10.0, random_state=42)
ransac.fit(X, Y)

# پیش‌بینی
Y_pred = ransac.predict(X)

# نقاط inlier و outlier را جدا کنیم
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

# رسم نمودار
plt.figure(figsize=(10, 5))
plt.scatter(X[inlier_mask], Y[inlier_mask], color='green', label='Inliers',
alpha=0.6)
plt.scatter(X[outlier_mask], Y[outlier_mask], color='red', label='Outliers',
alpha=0.6)
plt.plot(X, Y_pred, color='blue', label='RANSAC Regression')
plt.xlabel("Feature Index (ستون‌ها)")
plt.ylabel("Row 1 Value (مقدار)")
plt.title("Robust Regression on the first row of data using RANSAC")
plt.legend()
plt.grid(True)
plt.show()

# چاپ ضرایب مدل
intercept = ransac.estimator_.intercept_[0]
slope = ransac.estimator_.coef_[0][0]
print(f"✅ مدل رگرسیون مقاوم (RANSAC): y = {slope:.2f}x + {intercept:.2f}")
```

نتیجه کد برای سطر اول:

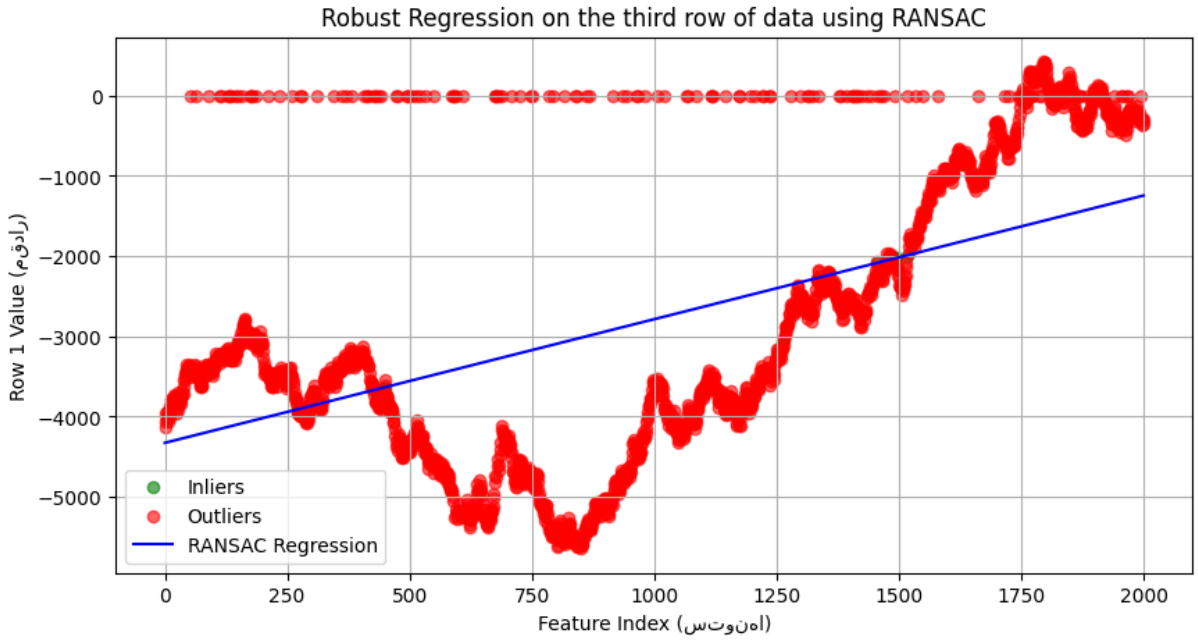


برای دیدن نتیجه کد برای سطر دوم و سوم همانند بخش قبلی فقط کافیست که در کد مقدار داخل [] را در دستور iloc به 1 برای ردیف دوم و 2 برای ردیف سوم عوض کنیم

نتیجه کد برای سطر دوم:



نتیجه کد برای سطر سوم:



همانطور که مشاهده میشود تاثیر داده های پرت در رگرسون مقاوم تقریبا دیده نمیشود و نتیجه رگرسیون دقیق تر بدست می آید و در این رگرسیون نیازی به پاکسازی داده ها نداشتیم و آن هم به دلیل عملکرد رگرسور ها بود که در تفاوت های اساسی به صورت زیر است

تفاوت روش RANSAC با Heuber در این است که در روش RANSAC مدل به زیر مجموعه ای از داده ها فیت می‌شود ولی در روش Huber تابع هزینه با توجه با مقدار خطا عوض میشود از طرفی مقاومت روش RANSAC به داده های پرت از Heuber خیلی بالاتر است و برای مواقعی که داده های پرت تعداد بالایی دارند مناسب تر است و در نهایت روش RANSAC داده های پرت را حذف کرده ولی ممکن است مدل ناپایدار شده یا نیاز به تنظیم دستی داشته باشد ولی روش Heuber ریاضی طور و روان تر است ولی در حضور داده های پرت زیاد ضعیف عمل می‌کند

2.4

جواب های این بخش در بخش های قبلی سوال ۲ همزمان انجام داده شده است

راهکار رگرسیون برای داده های ردیف سوم:

همانطور که دیده شد با رگرسیون خطی تحت همه روش های بالا داده های ردیف سوم به خوبی تقریب زده نمی‌شدند

استفاده از رگرسیون های دیگر مانند polynomial Regression که از یک منحنی از درجه دو یا سه یا بالاتر استفاده می‌کنه مثال اگر الگوی داده شبیه به $Y = aX^2 + bX + c$ باشد نمیشود آن را با یک خط مدل کرد ولی با منحنی راحت تر می‌شود

مدل های غیر خطی نیز وجود دارن که با توجه به الگوی بخش های مختلف نوع رگرسیون رو انتخاب میکنن مثل Decision Tree یاSVR

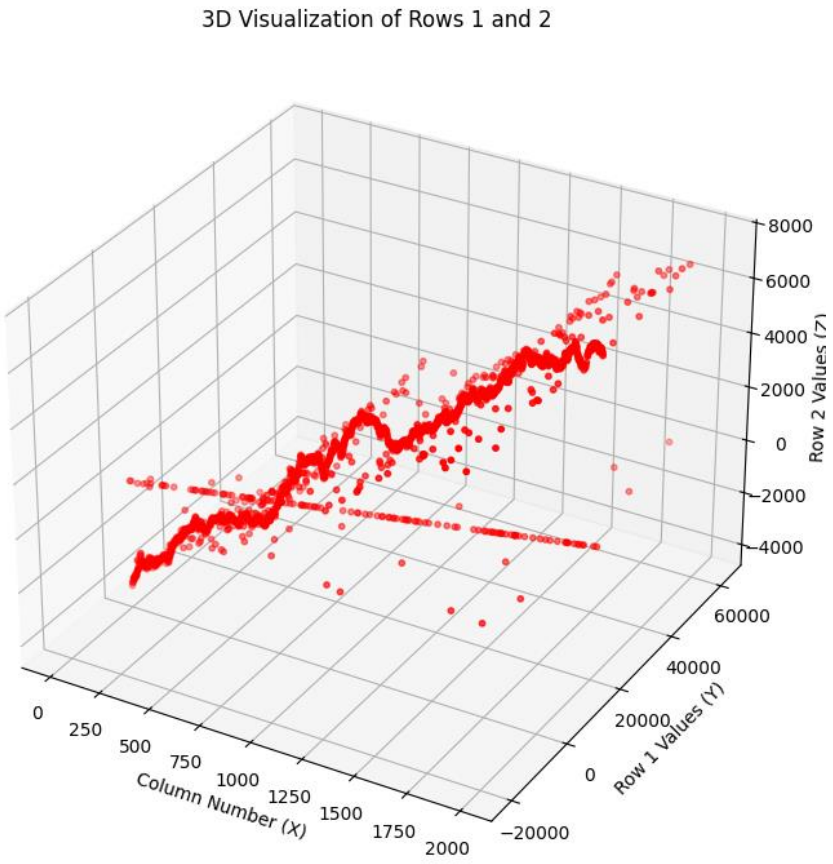
2.4.2

برای رسم داده ها در محیط سه بعدی با دیتاست ردیف ۱ و ۲ کافیست که داده هارا لود کرده با مشخص کردن ردیف اول و دوم آنها را به جای بردارهای Y و Z قرار دهیم و برای بردار X از ایندکس ستون ها استفاده کنیم

پس کد به صورت زیر خواهد بود

```
from mpl_toolkits.mplot3d import Axes3D

# Load your data
data = pd.read_csv("/content/mp1_lr_dataset_ai4032.csv")  # Make sure to update file
name if needed

# Extract rows
row1 = data.iloc[0].values  # Row 1 → Y-axis
row2 = data.iloc[1].values  # Row 2 → Z-axis
x = np.arange(1, len(row1) + 1)  # Column numbers → X-axis

# Create 3D plot
fig = plt.figure(figsize=(12, 7))
ax = fig.add_subplot(111, projection='3d')

# Plot the data
ax.scatter(x, row1, row2, c='red', marker='o', s=10)

# Set labels
ax.set_xlabel("Column Number (X)")
ax.set_ylabel("Row 1 Values (Y)")
ax.set_zlabel("Row 2 Values (Z)")
ax.set_title("3D Visualization of Rows 1 and 2")

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

نتیجه کد:



با توجه به شک امکان تقریب زدن داده ها با یک رگرسیون تک متغیره وجود دارد تا زمانی که هدف مدل کردن یکی از ردیف داده ها باشد مثلا فقط محور Y یا فقط محور Z ولی برای مدل کردن Z برحسب X و Y همزمان، دیگر امکان رگرسیون تک متغیره وجود ندارد و رگرسیون چند متغیره خواهد شد.

بخش امتیازی)

سوال ۱و۲

بررسی مفاهیم تابع هزینه و تابع اتلاف و تفاوت آن‌ها

در حوزه‌های مختلفی همچون یادگیری ماشین، علوم داده، و مهندسی کنترل، مفاهیم تابع اتلاف (Loss Function) و تابع هزینه (Cost Function) از اهمیت بالایی برخوردارند. این دو مفهوم گرچه در برخی متون به جای یکدیگر استفاده می‌شوند، اما در عمل دارای تفاوت‌های مهمی هستند. در این بخش به معرفی و مقایسه این دو تابع پرداخته می‌شود.

تابع اتلاف(Loss Function)

تابع اتلاف معیاری برای اندازه‌گیری میزان خطای پیش‌بینی مدل برای **یک نمونه داده** است. این تابع معمولاً ورودی‌های مدل (مقادیر پیش‌بینی‌شده) را با مقادیر واقعی مقایسه کرده و یک عدد به‌عنوان میزان خطا تولید می‌کند. هدف از تعریف تابع اتلاف، فراهم‌کردن معیاری برای ارزیابی عملکرد مدل در سطح نمونه‌ای است.

برای مثال در مسائل رگرسیون، یکی از رایج‌ترین توابع اتلاف، تابع میانگین مربعات خطا (MSE) است که برای یک نمونه به‌صورت زیر تعریف می‌شود:

$$\text{Loss}(y, \hat{y}) = (y - \hat{y})^2$$

در مسائل طبقه‌بندی، تابع اتلاف متقاطع (Cross-Entropy) از رایج‌ترین توابع مورد استفاده است.

تابع هزینه(Cost Function)

تابع هزینه، تعمیم‌یافته‌ی تابع اتلاف است و معمولاً به‌صورت میانگین یا مجموع خطاهای حاصل از تابع اتلاف روی **کل مجموعه داده‌ها** تعریف می‌شود. این تابع معیار اصلی برای ارزیابی عملکرد کلی مدل و بهینه‌سازی پارامترهای آن در فرآیند آموزش است.

تابع هزینه در واقع میانگینی از مقادیر تابع اتلاف برای تمام نمونه‌ها است:

$$\text{Cost} = (1/N) \sum_{i=1}^{N} \text{Loss}(y_i, \hat{y}_i)$$

که در آن N تعداد کل نمونه‌های آموزشی است.

تفاوت تابع هزینه و تابع اتلاف

ویژگی	(Loss Function) تابع اتلاف	(Cost Function) تابع هزینه
دامنه کاربرد	برای یک نمونه خاص	برای کل مجموعه داده‌ها
هدف	اندازه‌گیری خطا برای یک داده	ارزیابی کلی عملکرد مدل
استفاده	بررسی دقیق رفتار مدل روی یک نمونه	آموزش و بهینه‌سازی مدل
مقدار خروجی	خطای یک پیش‌بینی	میانگین یا مجموع خطاهای همه پیش‌بینی‌ها

جمع‌بندی

تابع اتلاف ابزاری برای سنجش دقت مدل در سطح خرد (یک نمونه) است، در حالی که تابع هزینه در سطح کلان (کل داده‌ها) عمل می‌کند و هدف اصلی در فرآیند آموزش مدل، کمینه‌سازی تابع هزینه است. در بسیاری از الگوریتم‌های یادگیری ماشین، ابتدا نوعی تابع اتلاف تعریف می‌شود و سپس با استفاده از آن، تابع هزینه برای کل مجموعه داده‌ها ساخته می‌شود. این دو مفهوم همچنین در مهندسی کنترل و بهینه‌سازی نیز کاربرد فراوانی دارند.

سوال ۳

به طور کلی مقدار مقدار واقعی را با Y_i و مقدار پیش بینی را با $\hat{Y}_i$ نشان میدهیم و اختلاف این دو همان خطای ما است یعنی [خطا $= Y_i - \hat{Y}_i$] و برای محاسبه این مقدار روش های مختلفی وجود دارند

روش اول روش میانگین مربعات خطا (MSE) است که رایج ترین نوع است که به دلیل اینکه خطا ها را به توان ۲ میرساند تاثیر بیشتری روی خطاهای بزرگتر میگذارد. ولی آموزش با این نوع سخنت تر است و فرمول آن به شرح زیر است

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |Y_i - \hat{Y}|$$

روش دوم Heuber Loss است مه ترکیبی از MSE و MAE است که همانطور که قبلا توضیح داده شده است برای خطاهای کوچک مثل MSE رفتار کرده و برای خطاهای بزرگ مثل MAE رفتار میکند این باعث می‌شود که نسبت به مقادیر outlier و پرت مقاوم تر رفتار کند ونتیجه بهتری به ما بدهد.

فرمول آن نیز به این صورت است:

$$L_{\delta}(Y,\hat{Y}) = \begin{cases} \frac{1}{2}(Y_i - \hat{Y})^2 & if\ |Y_i - \hat{Y}| \leq \delta \\ \delta.\left(|Y_i - \hat{Y}| - \frac{1}{2}\delta\right) & otherwise \end{cases}$$

روشی دیگر برای کمینه سازی خطاها روش RANSAC است که به جای کم کردن مجموع خطاها ابتدا یک زیر مجموعه تصادفی از داده ها یه نام inliner انتخاب می‌کند و سپس مدلی را با کترین خطا روس آن فیت می‌کند و از بین این زیرمجموعه ها مدلی که دارای بیشترین inliner است را انتخاب می‌کند، این کار باعث میشود تاثیر داده های پرت خیلی کم شود و رگرسیون مقاومت بالاتری نسبت به داده های پرت خواهد داشت و عمل فیت شدن روی داده های واقعی صورت میگیرد و پاسخ از تقریب بهتری برخوردار خواهد بود.

سوال ۴

برای داده هایی که در دسترس ما بود به دلیل اینکه تعداد داده ای پرت زیاد بودند عملکرد رگرسور های مقاوم Huber و RANSAC بهتر از MSE و یا MAE بودند و توانستند پاسخ دقیق تری بدهند.