Game Success Prediction

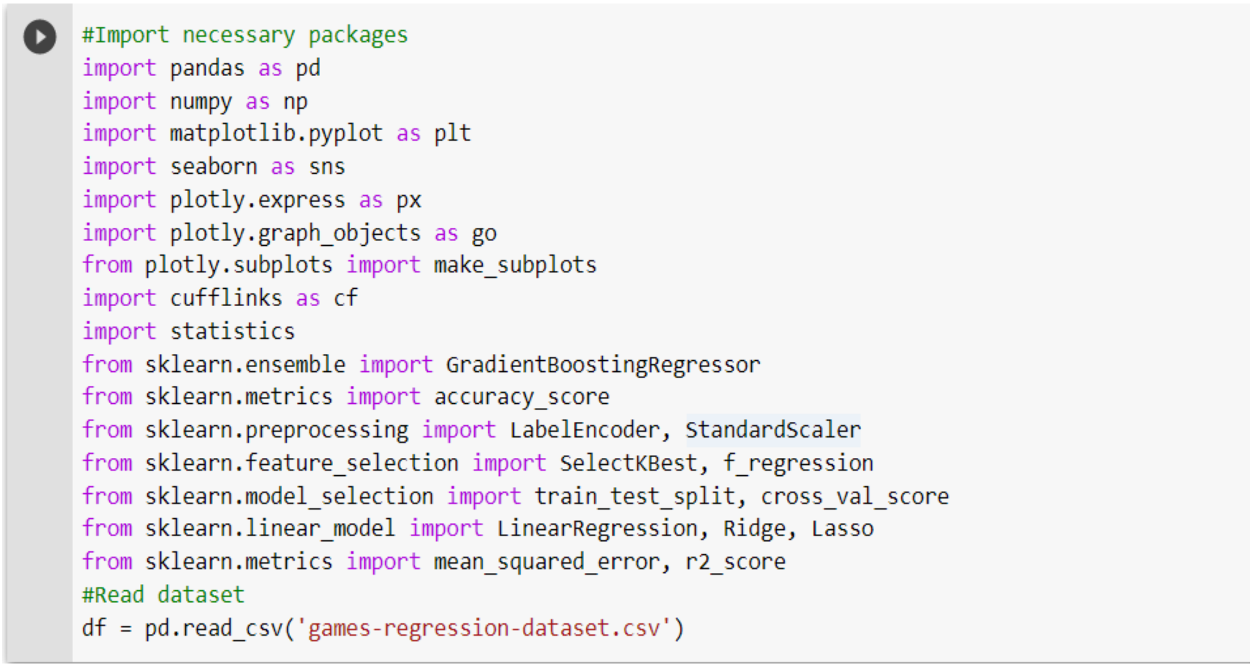
Software Engineering Project – Phase One

Team_ID: CS_27

عبد الرحمن فايز محمد حسن هندام	20201700479	Level 3 - cs
ماريا سامح صبحي مشرقي	20201700632	Level 3 - cs
محمد معوض عبده ذكي	20201700743	Level 3 - cs
مدحت عصام ابراهيم محمد علي	20201700789	Level 3 - cs
محمد ايمن رشدي عبد العظيم	20201700674	Level 3 - cs

Importing libraries:

The code imports various Visualizations & machine learning packages from scikit-learn, including seaborn , matplotlib, GradientBoostingRegressor, accuracy_score, LabelEncoder, StandardScaler, SelectKBest, f_regression, train_test_split, cross_val_score, LinearRegression, Ridge, Lasso, mean_squared_error, and r2_score. However, it does not utilize them in the current script.



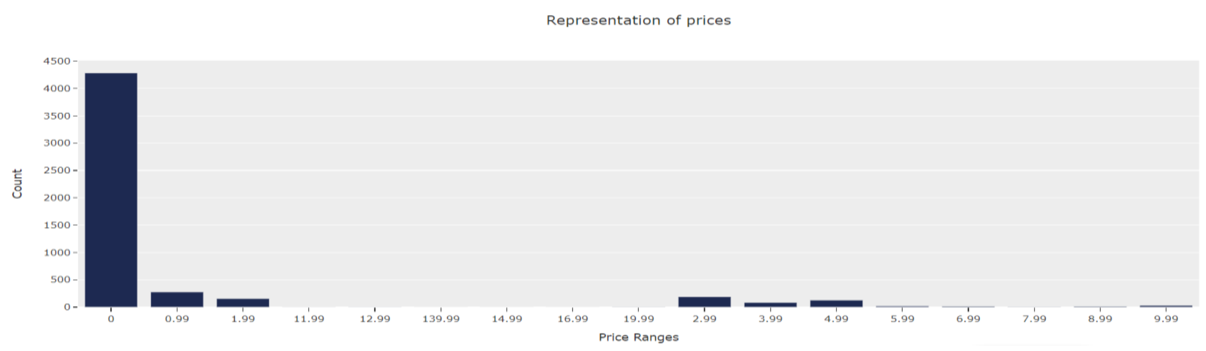
```python
#Import necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import cufflinks as cf
import statistics
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score
#Read dataset
df = pd.read_csv('games-regression-dataset.csv')
```

Data Visualization & Exploration:

The code is a Python script that imports necessary packages to perform exploratory data analysis and machine learning tasks. It reads a CSV file named 'games-regression-dataset.csv' using pandas library and assigns it to a Data Frame variable named 'df'. The shape of the Data Frame is (5214 rows, 18 attributes).

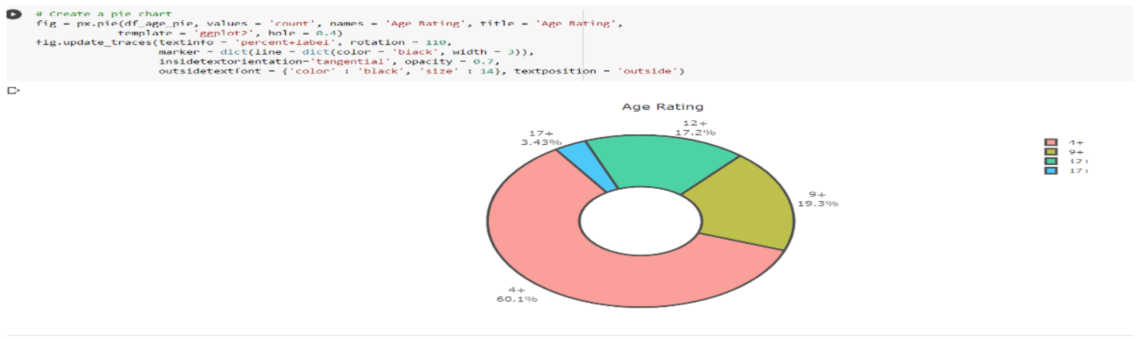
The script then proceeds to perform basic data analysis by calling various pandas Data Frame methods. It calculates the descriptive statistics of the 'Price', 'Size', and 'Average User Rating' columns using the 'describe' method and assigns it to the 'df_description' variable. It also prints the unique values in the 'Age Rating', 'Primary Genre', 'Languages', and 'Average User Rating' columns.

Graphical Representation for Games prices:



.

Age rating representation:



```
# Create a pie chart
fig = px.pie(df_age_pie, values = 'count', names = 'Age Rating', title = 'Age Rating',
             template = 'ggplot2', hole = 0.4)
fig.update_traces(textinfo = 'percent+label', rotation = 110,
                  marker = dict(line = dict(color = 'black', width = 3)),
                  insidetextorientation='tangential', opacity = 0.7,
                  outsidetextfont = {'color' : 'black', 'size' : 14}, textposition = 'outside')
```

Creating a new column which represent the number of languages each game acquire:

```
[22] # Define a function to count the number of languages in each record
     def count_languages(row):
         # Convert the Languages column to a string and split by commas
         languages = str(row['Languages']).split(',')
         # Count the number of languages and return the result
         return len(languages)

     # Apply the function to each row and create a new column called num_lang
     df['num_lang'] = df.apply(count_languages, axis=1)
```

Data preprocessing:

In this project, we will preprocess the dataset using several techniques. Firstly, we will remove duplicates and missing values, and we will encode categorical variables. Then, we will perform feature selection and scaling to prepare the data for the regression models. We will implement these techniques as follows:

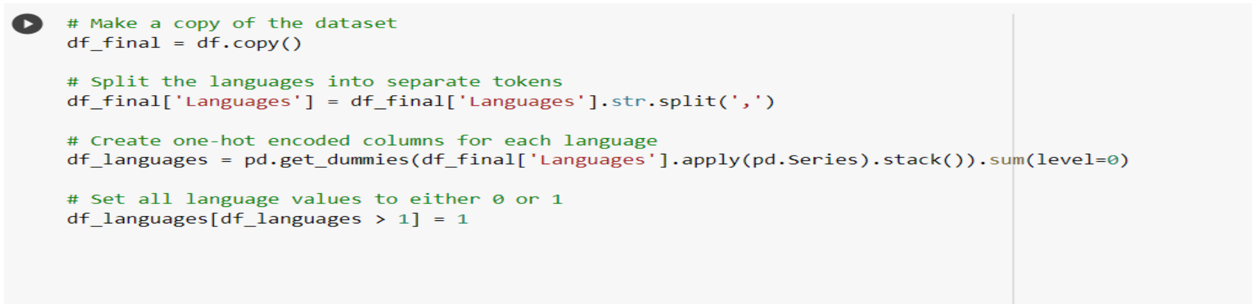
Removing duplicates and missing values:

We will check for duplicates using the drop_duplicates method in pandas, and we will remove missing values using the dropna method.

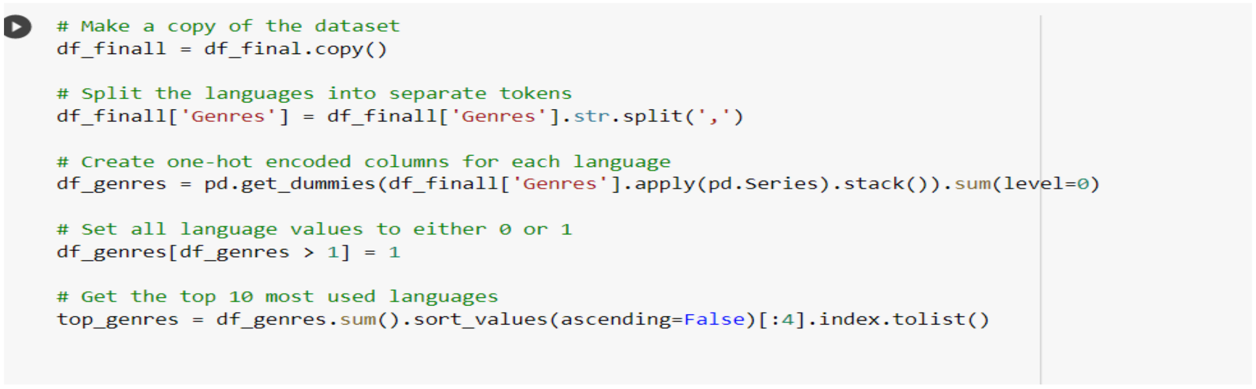
Encoding categorical variables:

We will use the LabelEncoder class from scikit-learn to encode the categorical variables 'Age Rating' and 'Primary Genre'.

Splitting Languages & Genres:



```
# Make a copy of the dataset
df_final = df.copy()

# Split the languages into separate tokens
df_final['Languages'] = df_final['Languages'].str.split(',')

# Create one-hot encoded columns for each language
df_languages = pd.get_dummies(df_final['Languages'].apply(pd.Series).stack()).sum(level=0)

# Set all language values to either 0 or 1
df_languages[df_languages > 1] = 1
```

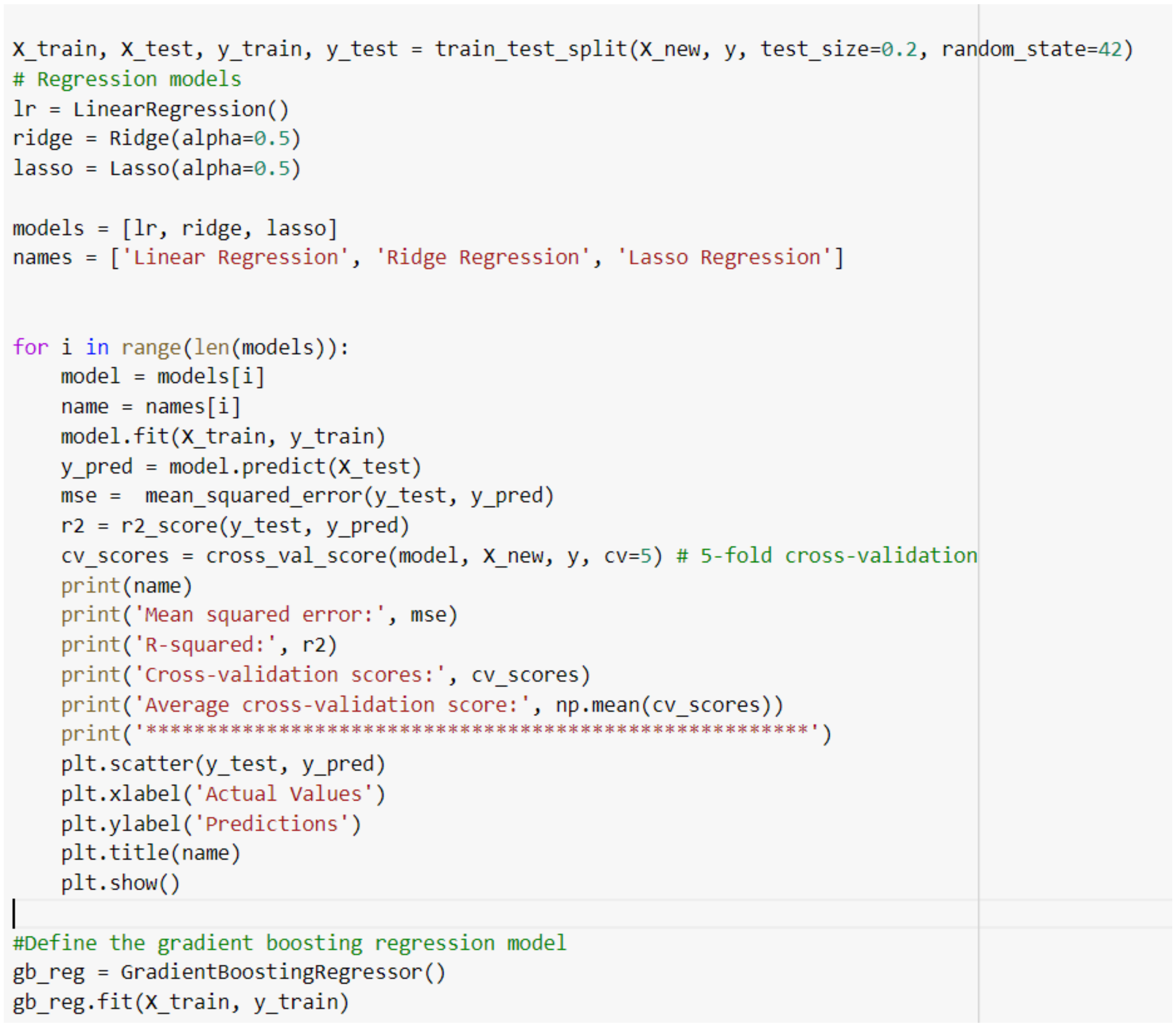


```
# Make a copy of the dataset
df_finall = df_final.copy()

# Split the languages into separate tokens
df_finall['Genres'] = df_finall['Genres'].str.split(',')

# Create one-hot encoded columns for each language
df_genres = pd.get_dummies(df_finall['Genres'].apply(pd.Series).stack()).sum(level=0)

# Set all language values to either 0 or 1
df_genres[df_genres > 1] = 1

# Get the top 10 most used languages
top_genres = df_genres.sum().sort_values(ascending=False)[:4].index.tolist()
```

Regression techniques & train test split:

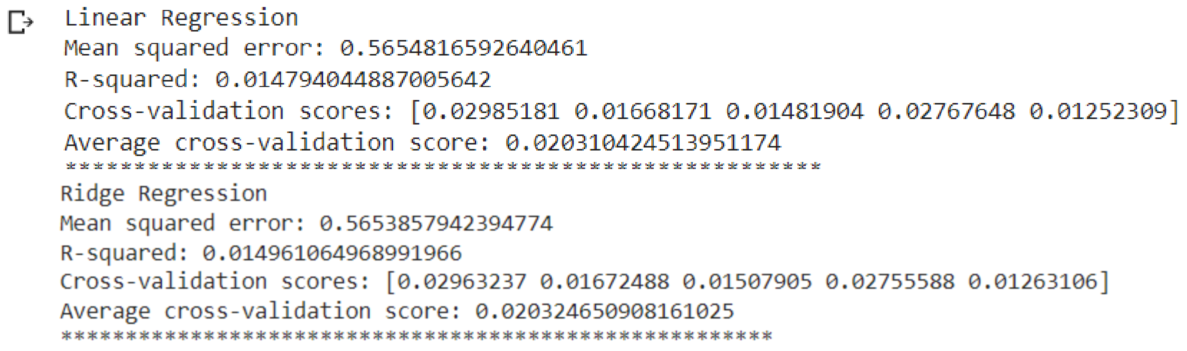
We will use multiple regression techniques to predict the 'Average User Rating' attribute: Linear Regression, Ridge Regression, Lasso Regression & gradient boosting regression model.

Linear Regression is a simple model that finds the best linear fit to the data, while Ridge Regression is a regularization technique that adds a penalty term to the loss function to prevent overfitting. We will compare the performance of the two models using the mean squared error and the R-squared score.

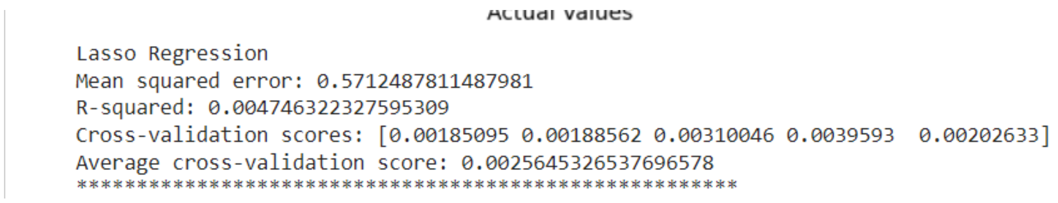


```
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2, random_state=42)
# Regression models
lr = LinearRegression()
ridge = Ridge(alpha=0.5)
lasso = Lasso(alpha=0.5)

models = [lr, ridge, lasso]
names = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']


for i in range(len(models)):
    model = models[i]
    name = names[i]
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse =  mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    cv_scores = cross_val_score(model, X_new, y, cv=5) # 5-fold cross-validation
    print(name)
    print('Mean squared error:', mse)
    print('R-squared:', r2)
    print('Cross-validation scores:', cv_scores)
    print('Average cross-validation score:', np.mean(cv_scores))
    print('*******************************************************')
    plt.scatter(y_test, y_pred)
    plt.xlabel('Actual Values')
    plt.ylabel('Predictions')
    plt.title(name)
    plt.show()

#Define the gradient boosting regression model
gb_reg = GradientBoostingRegressor()
gb_reg.fit(X_train, y_train)
```

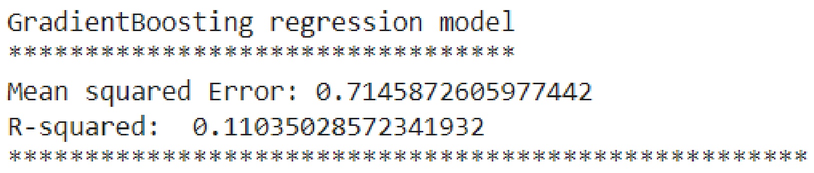
Regression Results:



```
Linear Regression
Mean squared error: 0.5654816592640461
R-squared: 0.014794044887005642
Cross-validation scores: [0.02985181 0.01668171 0.01481904 0.02767648 0.01252309]
Average cross-validation score: 0.020310424513951174
******************************************************
Ridge Regression
Mean squared error: 0.5653857942394774
R-squared: 0.014961064968991966
Cross-validation scores: [0.02963237 0.01672488 0.01507905 0.02755588 0.01263106]
Average cross-validation score: 0.020324650908161025
******************************************************
```

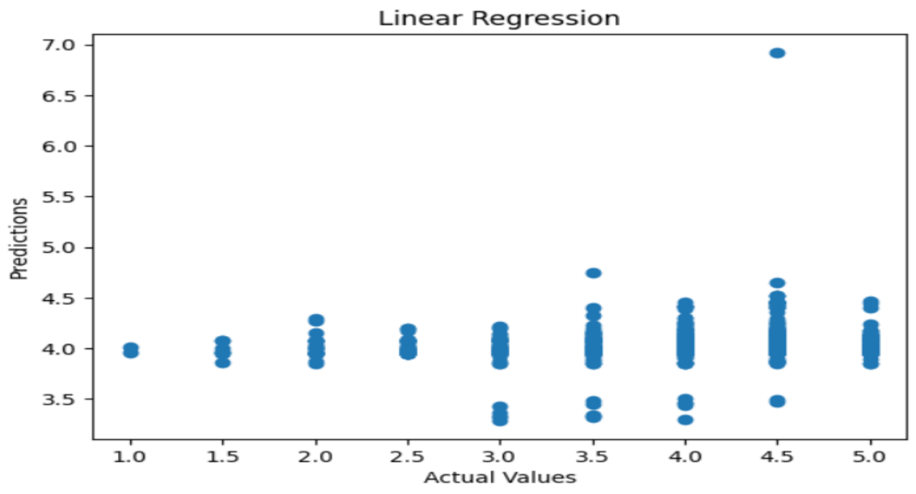


```
                         Actual values
Lasso Regression
Mean squared error: 0.5712487811487981
R-squared: 0.004746322327595309
Cross-validation scores: [0.00185095 0.00188562 0.00310046 0.0039593  0.00202633]
Average cross-validation score: 0.0025645326537696578
******************************************************
```



```
GradientBoosting regression model
**********************************
Mean squared Error: 0.7145872605977442
R-squared:  0.11035028572341932
******************************************************
```

Graphical representation for actual & predicted values of avg user rating using linear regression:


Linear Regression
Predictions
Actual Values