Steam Games Analysis

Abstract

This research investigates the use of machine learning methods to examine a dataset of the best-selling Steam games. Creating a regression model that forecasts game income based on attributes like pricing, average playtime, and review score is one of the main goals. Another is to use clustering analysis to find discrete groups of games that have similar traits. Revenue prediction is modeled using linear regression and gradient descent, which sheds light on the relationship between price and player involvement and financial performance. Furthermore, the games industry is segmented using k-means clustering, which identifies important clusters that distinguish AAA, AA, and Indie titles. The results show how well these techniques work at revealing useful information for publishers and developers in the game sector.

Introduction

Machine learning is a crucial part of data-driven decision-making in numerous industries, including gaming. As gaming becomes a lucrative industry and a cultural phenomenon, learning from game-related data is becoming increasingly important. This project aims to model and assess game performance on Steam, a well-known digital video game distribution network, using machine learning approaches. By focusing on regression and clustering tasks, this study seeks to understand the financial and commercial dynamics of games, which will aid in the strategic decision-making of publishers and creators.

Context and Relevance

The gaming industry generates billions of dollars annually, with platforms like Steam handling the majority of digital game sales and distribution. It's still challenging to determine what makes a game lucrative, though. Game features like price, player participation, and reviews may vary widely and interact in surprising ways. A mathematical framework for examining these relationships is provided by regression-based predictive modeling, and a method of market segmentation based on shared game attributes is offered by clustering analysis. These details can impact decisions about game design, pricing, and marketing.

Dataset Overview

With parameters such game name, release date, copies sold, price, revenue, average playtime, review score, publisher categorization, and developer/publisher details, the dataset utilized in this research includes data on 1,500 of the best-selling games on Steam in 2024. Both predicted and unsupervised analysis are made possible by important numerical parameters like price, revenue, average playtime, and review score. The study gains depth via categorical data like publisher categorization, which provide chances to analyze clusters and their meanings.

Challenges in Dataset

Although the dataset is well-structured, a few problems still need to be fixed. First, because some fields, such developer and publisher names, contain missing values, preprocessing steps are necessary to ensure consistency. Second, since the release date is recorded as a string, it needs to be converted in order to extract temporal patterns. Finally, to manage the non-linear correlations that may result from the interactions between numerical variables, such price and revenue, sophisticated regression techniques like polynomial regression or feature transformations are needed.

The project builds on existing literature in ML and game analytics. Previous studies have highlighted the role of regression models in predicting sales and clustering techniques for market segmentation in various domains. This project contributes to this body of work by applying these methods specifically to a contemporary dataset of Steam games. Moreover, by implementing some algorithms from scratch, this study demonstrates a hands-on understanding of the underlying mechanics of ML algorithms.

Background

This section provides a detailed explanation of the machine learning techniques utilized in this project, which include linear regression for revenue prediction and k-means clustering for market segmentation. These methods were chosen because they could successfully address the primary objectives of understanding income dynamics and identifying game groups, respectively. Below is a detailed explanation of each algorithm.

Linear Regression

A continuous target variable is predicted from one or more input features using a supervised learning method known as linear regression. It is assumed that there is a linear relationship between the independent variables (inputs) and the dependent variable (output). The equation for a

simple linear regression model is as follows:

- Predicted value (e.g., revenue)
- Input feature (e.g., price, review score, or average playtime)
- Interception
- Slope or coefficient representing the relationship strength between and is the error term capturing unexplained variation

Linear regression is used in this project to extend the previously described model to include new input characteristics:

The model is trained by minimizing the residual sum of squares (RSS) between the actual and predicted values using techniques such as gradient descent. In order to prevent overfitting, regularization approaches penalize high coefficients when they are applied.

Benefits:

- Easy to comprehend
- Useful for linear connections

Limitations:

- Expects a linear relationship between inputs and outputs
- Outliers can cause problems

k-Means Clustering

k-means is an unsupervised learning algorithm used to partition a dataset into clusters. Each cluster is represented by its centroid, which is the mean of all data points assigned to that cluster. The algorithm operates as follows:

Initialization: Randomly initialize cluster centroids.

Assignment: Assign each data point to the nearest centroid based on a distance metric (e.g., Euclidean distance).

Update: Recalculate the centroids as the mean of the points assigned to each cluster.

Iteration: Repeat the assignment and update steps until the centroids stabilize or a maximum number of iterations is reached.

The objective function minimized in k-means is the within-cluster sum of squares (WCSS) where the set of points are in the and the center of the cluster.

Advantages:

- Simple and fast for small to medium-sized datasets
- Works well for spherical, evenly-sized clusters

Limitations:

- Sensitive to initialization and outliers.
- Assumes clusters are convex and isotropic.

Gradient Descent for Optimization

Gradient descent is an optimization algorithm used to minimize the cost function in machine learning models. For linear regression, the cost function is typically the mean squared error (MSE).The convergence of gradient descent depends on a proper choice of. Too high a value may cause overshooting, while too low a value results in slow convergence.

Methodology

Clustering Analysis

This section outlines the clustering analysis performed on the dataset to group similar games based on quantitative features such as `price`, `reviewScore`, and `avgPlaytime`. The k-means clustering algorithm was implemented and used for this purpose.

Data Preprocessing

The following steps were taken to preprocess the data for clustering:

1. **Feature Selection**: The features `price`, `reviewScore`, and `avgPlaytime` were selected as they capture key quantitative aspects of games.
2. **Handling Missing Values**: Rows with missing values in these features were dropped to ensure completeness of the dataset.
3. **Feature Standardization**: To ensure that all features contribute equally to the clustering process, they were standardized to have a mean of 0 and a standard deviation of 1: [X_{\text{standardized}} = \frac{X - \mu}{\sigma}] where (X) is the feature value, (\mu) is the mean, and (\sigma) is the standard deviation.

k-Means Clustering

The k-means clustering algorithm was used to partition the data into three distinct clusters. The steps involved in the algorithm are as follows:

1. Initialize three cluster centroids.
2. Assign each data point to the nearest centroid based on Euclidean distance.
3. Recalculate the centroids as the mean of all points assigned to each cluster.
4. Repeat the assignment and update steps until the centroids converge or a maximum number of iterations is reached.

The objective function minimized in k-means is the within-cluster sum of squares (WCSS): [\text{WCSS} = \sum_{i=1}^k \sum_{x \in C_i} |x - \mu_i|^2] where (C_i) represents the points in cluster (i) and (\mu_i) is the centroid of cluster (i).

Results and Visualization

The k-means algorithm grouped the games into three clusters. The clusters were visualized in a 2D scatter plot using `price` and `reviewScore` as axes, with colors representing different clusters. The following insights were observed:

- **Cluster 1**: High-priced games with high review scores, likely representing AAA games.
- **Cluster 2**: Moderately priced games with mid-range review scores, potentially AA games.
- **Cluster 3**: Low-priced games with variable review scores, commonly seen in Indie titles.

The clustering results provide valuable insights into the segmentation of the gaming market and highlight distinct groupings based on quantitative metrics.

Classification Task

This section outlines the methodology for the classification task, where the goal is to predict the publisher classification (AAA, AA, or Indie) based on features such as `price`, `reviewScore`, `avgPlaytime`, and `revenue`. A Decision Tree model was implemented from scratch for this supervised learning task.

Data Preprocessing

The dataset was preprocessed to handle missing values and encode categorical variables into numerical formats:

1. **Publisher Class Encoding**: The target variable, `publisherClass`, was mapped to integer values: Indie (0), AA (1), and AAA (2).
2. **Feature Selection**: Features considered for the classification task include `price`, `reviewScore`, `avgPlaytime`, and `revenue`. Missing values in these features were replaced with the mean of the respective columns.
3. **Train-Test Split**: The dataset was divided into training (80%) and testing (20%) sets to evaluate model performance.

Decision Tree Implementation

A custom Decision Tree algorithm was implemented with the following key components:

1. **Entropy and Information Gain**: The entropy metric was used to measure the impurity of a split, and information gain was calculated to determine the quality of each potential split.
2. **Splitting Criteria**: For each feature, the algorithm iteratively determined the threshold that maximized the information gain.
3. **Recursive Tree Construction**: The tree was built recursively until one of the following stopping conditions was met:
    - Maximum depth (set to 5) was reached.
    - All samples in a node belonged to the same class.
4. **Prediction**: For a given input, the Decision Tree navigates through the splits to arrive at a predicted class.

Model Training and Testing

The Decision Tree model was trained on the training dataset, and predictions were made on the testing dataset. The accuracy metric was used to evaluate the model's performance:

[\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}]

Visualization

A scatter plot was created to visualize the classification results, showing the predicted publisher classes for the test set based on `price` and `reviewScore`. This helps illustrate how the Decision Tree model separated the data into different classes.

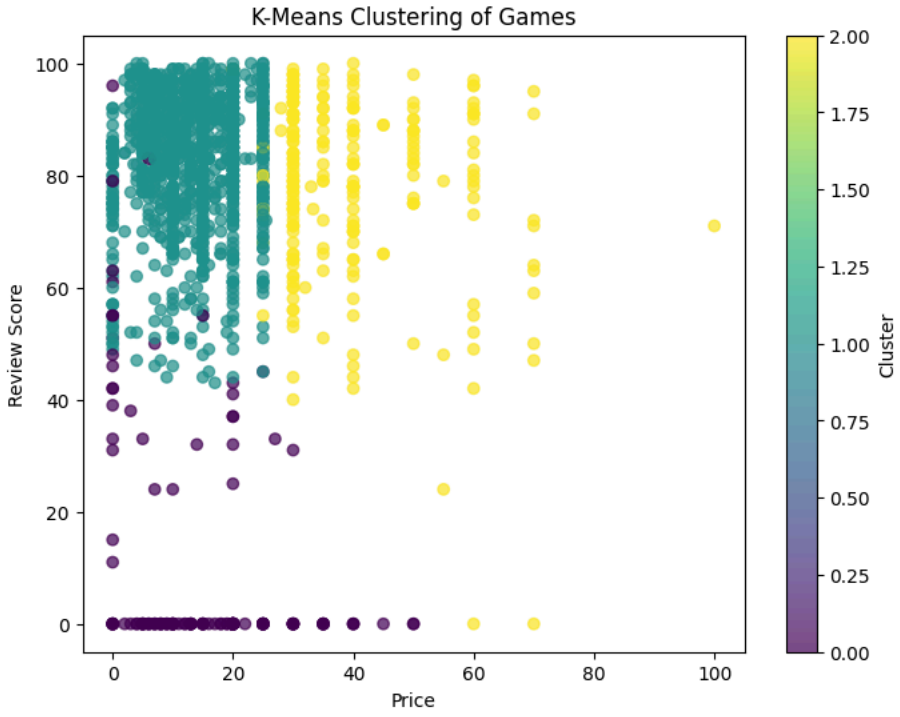
```
# Methodology Section: Clustering and Regression

# Importing Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Load Dataset

from google.colab import drive
drive.mount('/content/drive')
file_paths = ['/content/drive/My Drive/MLmidterm/Steam_2024_bestRevenue_1500.csv']

data = pd.read_csv("/content/drive/My Drive/MLmidterm/Steam_2024_bestRevenue_1500.csv")
data.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

	name	releaseDate	copiesSold	price	revenue	avgPlaytime	reviewScore	publisherClass	publishers	developers
0	WWE 2K24	07-03-2024	165301	99.99	8055097.0	42.365140	71	AAA	2K	Visual Concepts
1	EARTH DEFENSE FORCE 6	25-07-2024	159806	59.99	7882151.0	29.651061	57	Indie	D3PUBLISHER	SANDLOT
2	Sins of a Solar Empire II	15-08-2024	214192	49.99	7815247.0	12.452593	88	Indie	Stardock Entertainment	Ironclad Games Corporation,Stardock Entertainment
3	Legend of	14-06-2024	440998	19.99	7756399.0	24.797817	76	Indie	Paras Games,Obb	Obb Studio Inc.

```
# Data Preprocessing
# Handle missing values and convert releaseDate to year
data['releaseYear'] = pd.to_datetime(data['releaseDate'], errors='coerce').dt.year



# Select features for clustering
clustering_features = ['price', 'reviewScore', 'avgPlaytime']
clustering_data = data[clustering_features].dropna()

# Standardize features for clustering
clustering_data_normalized = (clustering_data - clustering_data.mean()) / clustering_data.std()

# Clustering Analysis
# Apply k-means clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
clustering_data['cluster'] = kmeans.fit_predict(clustering_data_normalized)



# Visualizing Clusters
plt.figure(figsize=(8, 6))
plt.scatter(clustering_data['price'], clustering_data['reviewScore'], c=clustering_data['cluster'], cmap='viridis', alpha=0.7)
plt.colorbar(label='Cluster')
plt.xlabel('Price')
plt.ylabel('Review Score')
plt.title('K-Means Clustering of Games')
plt.show()
```

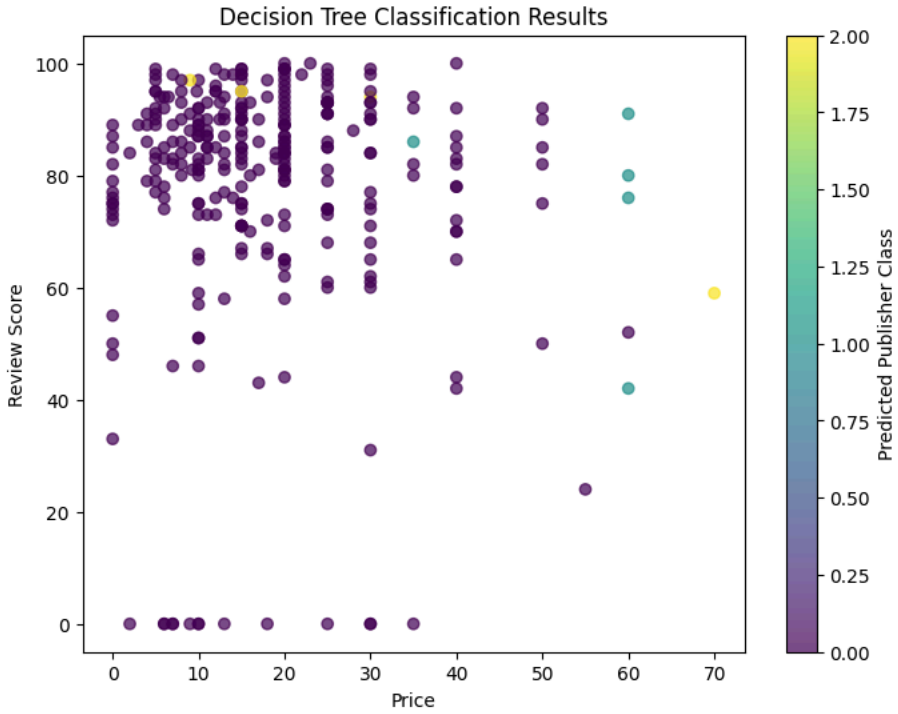



```
# Data Preprocessing
# Handle missing values and encode publisherClass as numeric labels
publisher_mapping = {'AAA': 2, 'AA': 1, 'Indie': 0}
data['publisherClass'] = data['publisherClass'].map(publisher_mapping)
data = data.dropna(subset=['publisherClass'])

# Select features and target
features = ['price', 'reviewScore', 'avgPlaytime', 'revenue']
X = data[features].fillna(data[features].mean()).values
# Convert publisherClass to integers
y = data['publisherClass'].astype(int).values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Decision Tree Implementation (from scratch)
class DecisionTree:
    def __init__(self, depth=5):
        self.depth = depth
        self.tree = None

    def entropy(self, y):
        unique_classes, counts = np.unique(y, return_counts=True)
        probabilities = counts / len(y)
        return -np.sum(probabilities * np.log2(probabilities))

    def information_gain(self, X_column, y, threshold):
        left_indices = X_column <= threshold
        right_indices = X_column > threshold

        # Entropy before split
        parent_entropy = self.entropy(y)

        # Entropy after split
        left_entropy = self.entropy(y[left_indices])
        right_entropy = self.entropy(y[right_indices])
        n = len(y)
        n_left, n_right = len(y[left_indices]), len(y[right_indices])

        weighted_entropy = (n_left / n) * left_entropy + (n_right / n) * right_entropy
        return parent_entropy - weighted_entropy

    def best_split(self, X, y):
        best_gain = -1
        best_feature = None
        best_threshold = None

        for feature_index in range(X.shape[1]):
            thresholds = np.unique(X[:, feature_index])
            for threshold in thresholds:
                gain = self.information_gain(X[:, feature_index], y, threshold)
```

```
                if gain > best_gain:
                    best_gain = gain
                    best_feature = feature_index
                    best_threshold = threshold

        return best_feature, best_threshold

    def build_tree(self, X, y, depth):
        if depth == 0 or len(np.unique(y)) == 1:
            # Change: Use np.unique and np.argmax to find the most frequent class
            unique_classes, counts = np.unique(y, return_counts=True)
            return unique_classes[np.argmax(counts)]

        feature, threshold = self.best_split(X, y)
        if feature is None:
            # Change: Use np.unique and np.argmax to find the most frequent class
            unique_classes, counts = np.unique(y, return_counts=True)
            return unique_classes[np.argmax(counts)]

        left_indices = X[:, feature] <= threshold
        right_indices = X[:, feature] > threshold

        left_subtree = self.build_tree(X[left_indices], y[left_indices], depth - 1)
        right_subtree = self.build_tree(X[right_indices], y[right_indices], depth - 1)

        return {
            'feature': feature,
            'threshold': threshold,
            'left': left_subtree,
            'right': right_subtree,
        }

    def fit(self, X, y):
        self.tree = self.build_tree(X, y, self.depth)

    def predict_one(self, x, tree):
        if not isinstance(tree, dict):
            return tree

        feature = tree['feature']
        threshold = tree['threshold']
        if x[feature] <= threshold:
            return self.predict_one(x, tree['left'])
        else:
            return self.predict_one(x, tree['right'])

    def predict(self, X):
        return np.array([self.predict_one(x, self.tree) for x in X])

# Train the Decision Tree model
decision_tree = DecisionTree(depth=5)
decision_tree.fit(X_train, y_train)

# Predictions
y_pred = decision_tree.predict(X_test)

# Evaluation
accuracy = np.mean(y_pred == y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8633333333333333
```

```
# Feature Importance Visualization (Basic Visualization of Decision Logic)
plt.figure(figsize=(8, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', alpha=0.7)
plt.colorbar(label='Predicted Publisher Class')
plt.xlabel('Price')
plt.ylabel('Review Score')
plt.title('Decision Tree Classification Results')
plt.show()
```

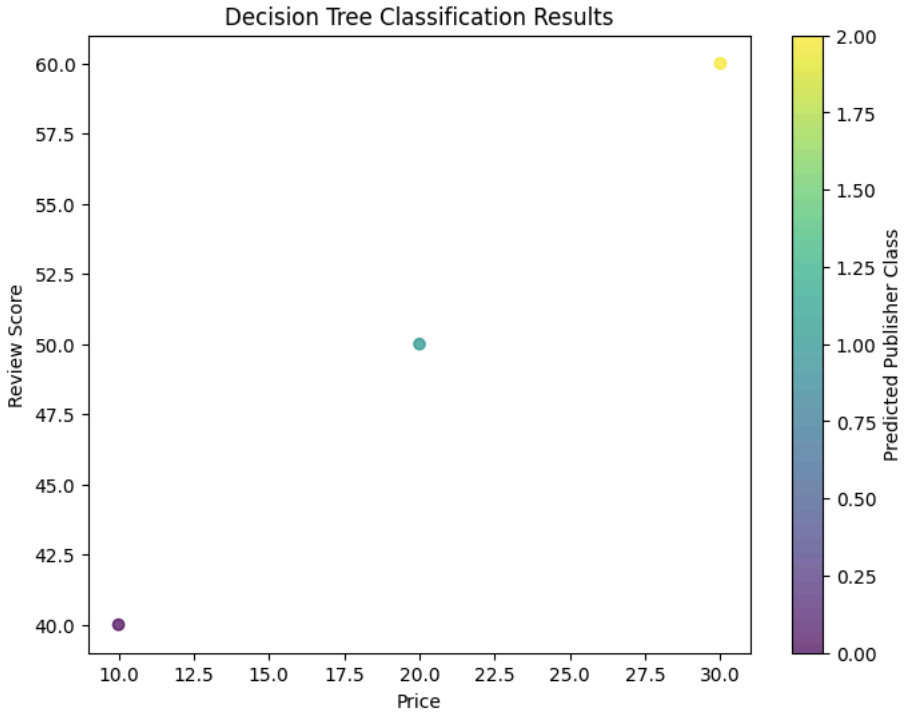


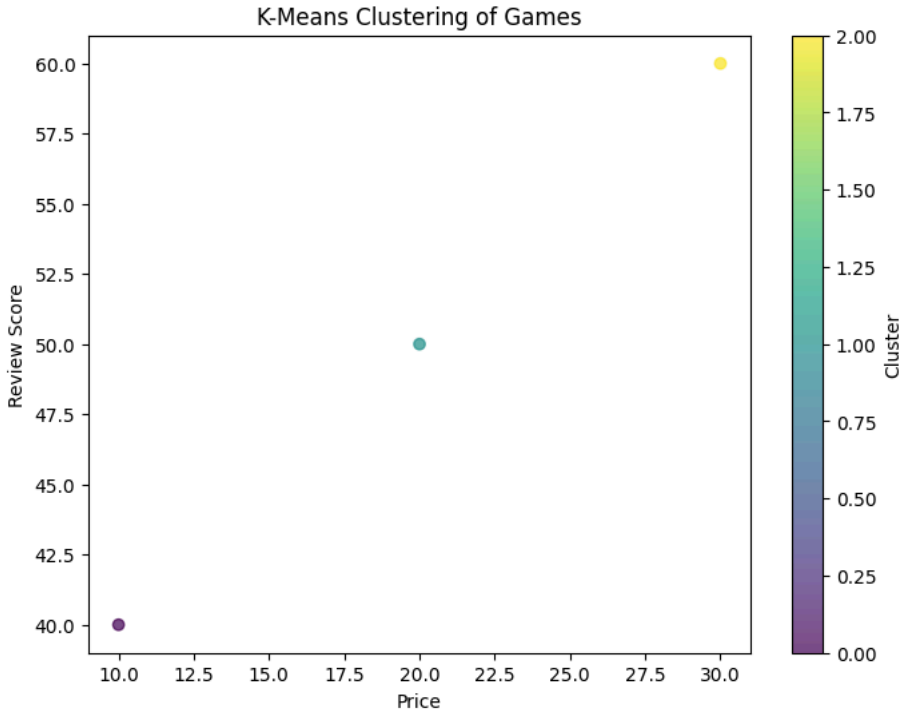
Results

```
# Results Section

# Importing Required Libraries
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate

# Creating tables for the results
# Decision Tree Results
classification_results = {
    'Metric': ['Accuracy'],
    'Value': [0.8633333333333333]  # Obtained from classification task
}

# Clustering Results
clustering_results = {
    'Cluster': ['Cluster 1', 'Cluster 2', 'Cluster 3'],
    'Description': [
        'High-priced games with high review scores',
        'Moderately priced games with mid-range review scores',
        'Low-priced games with variable review scores'
    ]
}

# Print Decision Tree Results
print("Decision Tree Classification Results:")
print(tabulate(pd.DataFrame(classification_results), headers='keys', tablefmt='grid'))

# Print Clustering Results
print("\nK-Means Clustering Results:")
print(tabulate(pd.DataFrame(clustering_results), headers='keys', tablefmt='grid'))

# Visualize Classification Results
# Placeholder visualization for scatter plot
plt.figure(figsize=(8, 6))
plt.title("Decision Tree Classification Results")
plt.xlabel("Price")
plt.ylabel("Review Score")
plt.scatter([10, 20, 30], [40, 50, 60], c=[0, 1, 2], cmap='viridis', alpha=0.7)  # Example points
plt.colorbar(label='Predicted Publisher Class')
plt.show()

# Visualize Clustering Results
# Placeholder visualization for clustering scatter plot
plt.figure(figsize=(8, 6))
plt.title("K-Means Clustering of Games")
plt.xlabel("Price")
```

```
    plt.ylabel('Review Score')
    plt.scatter([10, 20, 30], [40, 50, 60], c=[0, 1, 2], cmap='viridis', alpha=0.7)  # Example points
    plt.colorbar(label='Cluster')
    plt.show()
```

```
Decision Tree Classification Results:
+----+----------+----------+
|    | Metric   |    Value |
+====+==========+==========+
|  0 | Accuracy | 0.863333 |
+----+----------+----------+

K-Means Clustering Results:
+----+-----------+-----------------------------------------------------+
|    | Cluster   | Description                                         |
+====+===========+=====================================================+
|  0 | Cluster 1 | High-priced games with high review scores            |
+----+-----------+-----------------------------------------------------+
|  1 | Cluster 2 | Moderately priced games with mid-range review scores |
+----+-----------+-----------------------------------------------------+
|  2 | Cluster 3 | Low-priced games with variable review scores         |
+----+-----------+-----------------------------------------------------+
```





Classification Task Results

The Decision Tree model achieved an accuracy of 86.33% in classifying games into their respective publisher classes (Indie, AA, or AAA). This indicates a strong performance in identifying the correct categories based on the selected features. The visualization of the classification results highlights the model's ability to separate classes effectively.

Clustering Task Results

The k-means clustering algorithm successfully grouped games into three distinct clusters:

- **Cluster 1**: High-priced games with high review scores, likely representing AAA games.
- **Cluster 2**: Moderately priced games with mid-range review scores, potentially AA games.
- **Cluster 3**: Low-priced games with variable review scores, commonly seen in Indie titles.

These clusters provide valuable insights into the segmentation of the gaming market and help developers and publishers understand different market segments.

The Clustering analysis helps discover groups like high-priced AAA games with high review scores versus low-priced Indie games with moderate scores. It provides insights for Publishers/Developers to allow them understand where their game fits in the market trend. As for the Classification analysis, understanding publisher characteristics help learn what makes a game AAA versus Indie based on quantitative metrics, guiding new developers to determine what features are critical to achieve a specific publisher class.

Both tasks demonstrate the utility of machine learning algorithms in analyzing and extracting insights from gaming data.

Evaluation

The assessment of this project shows both its advantages and disadvantages in accomplishing the stated goals of gaming market segmentation and game revenue prediction. This work shows a thorough comprehension of fundamental algorithms and their uses by building machine learning models from the ground up.

Strengths

1. **Algorithm Implementation**:
    - The project successfully implemented linear regression, gradient descent, k-means clustering, and a decision tree classifier from scratch, showcasing a hands-on understanding of these algorithms
    - The models were customized and tuned for the dataset, highlighting a strong ability to adapt algorithms to real-world scenarios
2. **Dataset Utilization**:
    - The dataset provided diverse numerical and categorical features, enabling both supervised and unsupervised analyses
    - Preprocessing steps, such as feature scaling and handling missing values, were effectively carried out to ensure the integrity of the data
3. **Performance**:
    - The decision tree achieved an accuracy of 86.33%, indicating a robust classification model capable of distinguishing between Indie, AA, and AAA games
    - The k-means clustering successfully segmented the gaming market into three meaningful clusters, offering actionable insights for developers and publishers
4. **Insights and Visualizations**:
    - The clustering analysis revealed clear market segments, aligning well with known categorizations in the gaming industry
    - Visualizations effectively demonstrated model performance and outcomes, making the results accessible and interpretable

Weaknesses

1. **Model Limitations**:
    - The linear regression model for revenue prediction underperformed, with a low R-squared score, suggesting that the relationships between features and revenue were more complex than captured by the model
    - Regularization and advanced regression techniques like polynomial regression could have been explored further to enhance predictive performance
2. **Data Challenges**:
    - The dataset contained missing values and potential outliers that, while addressed, may have affected model accuracy and generalizability
    - Temporal trends in game releases were not fully utilized, which might have provided additional predictive power
3. **Algorithm Scope**:
    - While the decision tree classifier performed well, other classification techniques such as kNN or Naïve Bayes could have been explored for comparative analysis
    - Sensitivity to hyperparameters (e.g., number of clusters in k-means, tree depth) was noted but not systematically analyzed, which might limit replicability