Experiment 8

Aim: To implement a recommendation system on your dataset using the following machine learning techniques: Regression, Classification, Clustering, Decision tree, Anomaly detection, Dimensionality Reduction, Ensemble Methods

Theory:

1. Regression : Regression is a supervised learning technique used to predict continuous values based on input features. It estimates the relationship between dependent and independent variables.

- **Linear Regression**: Models a straight-line relationship between variables.
- **Polynomial Regression**: Captures non-linear relationships by introducing polynomial terms.
- **Ridge/Lasso Regression**: Regularized versions that prevent overfitting.
- **Logistic Regression**: Used for classification despite the name "regression."

2. Classification: Classification is a supervised learning technique that categorizes input data into predefined classes or labels.

- **Binary Classification**: Two possible outcomes (e.g., spam vs. not spam).
- **Multiclass Classification**: More than two categories (e.g., classifying animals as cat, dog, or bird).
- **Popular Algorithms**: Logistic Regression, Decision Trees, SVM, Random Forest, Neural Networks.

3. Clustering: Clustering is an **unsupervised learning** technique used to group similar data points together based on patterns. Unlike classification, clusters are not predefined.

- **K-Means**: Partitions data into K clusters using centroids.
- **Hierarchical Clustering**: Forms a tree-like structure of nested clusters.
- **DBSCAN**: Groups based on density, identifying outliers as noise.

4. Decision Tree: A Decision Tree is a tree-like structure where data is split into branches based on feature values. It is used for both classification and regression.

5. Anomaly Detection: Anomaly detection identifies unusual patterns that deviate significantly from normal data. It is widely used in fraud detection, cybersecurity, and medical diagnosis.

- **Statistical Methods**: Z-score, Gaussian distribution analysis.
- **Machine Learning Methods**: Isolation Forest, One-Class SVM, Autoencoders.
- **Distance-Based Methods**: k-Nearest Neighbors (k-NN) for detecting outliers.

6. Dimensionality Reduction: Dimensionality reduction is used to **reduce the number of input features** while preserving essential information. This helps improve model efficiency and visualization.

- **Principal Component Analysis (PCA)**: Converts correlated features into uncorrelated principal components.
- **t-SNE (t-Distributed Stochastic Neighbor Embedding)**: Useful for visualizing high-dimensional data.
- **Autoencoders**: Neural networks that learn compressed representations.

7. Ensemble Methods

Ensemble methods combine multiple models to improve accuracy and robustness. They work by aggregating predictions from multiple weak learners.

- **Bagging (Bootstrap Aggregating)**: Example: Random Forest (uses multiple decision trees).
- **Boosting**: Example: AdaBoost, XGBoost (sequentially improves weak models).
- **Stacking**: Combines multiple models using another model (meta-learner) to make final predictions.

Implementation:

Loading the dataset

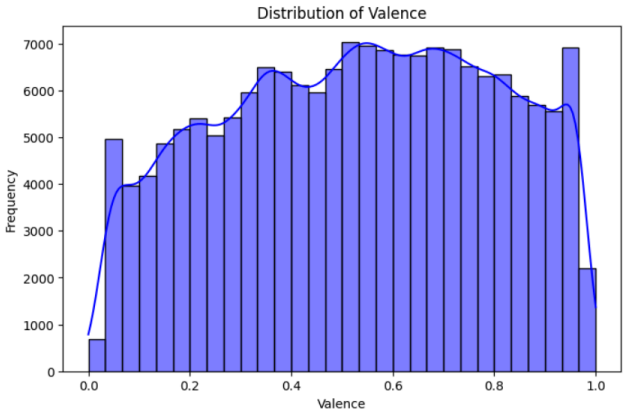
```
import pandas as pd
# Load the dataset
df = pd.read_csv("data.csv")
# Display first few rows
df.head()
```

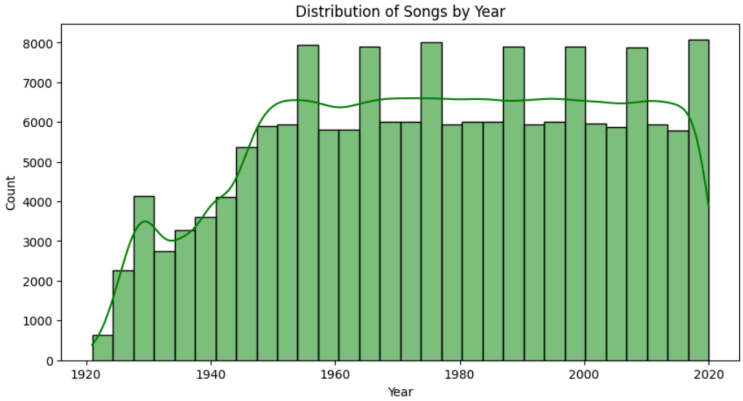
	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	liveness
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878000	10	0.665
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000000	7	0.160
2	0.0394	1921	0.961	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.166	0	1o6I8BglA6ylDMrIELygv1	0.913000	3	0.101

```
# Check for missing values
df.isnull().sum()
# Summary statistics
df.describe()
# Check data types
df.dtypes
```

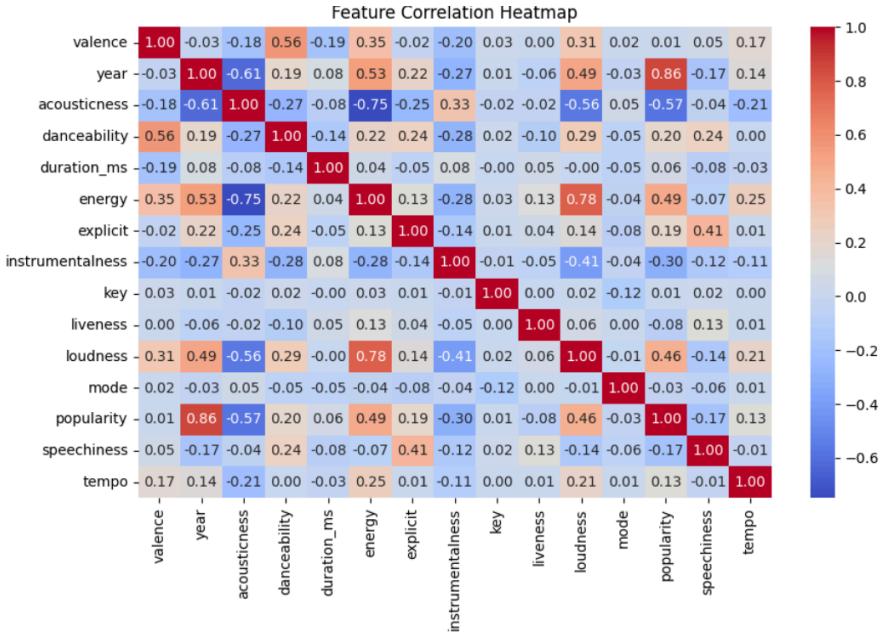
	0
valence	float64
year	int64
acousticness	float64
artists	object
danceability	float64
duration_ms	int64
energy	float64
explicit	int64

Visualisation





Heatmap



1. Regression

```
features = ["danceability", "energy", "loudness", "speechiness", "acousticness", "instrumentalness", "tempo","year"]
target = "popularity"
X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Mean Squared Error: 115.41
R² Score: 0.76
```

- **Columns Used:** The regression model uses features with high correlation to popularity, including danceability, energy, loudness, speechiness, acousticness, instrumentalness, tempo, and year. These features strongly influence a song's popularity.
- **Approach**: A Linear Regression model was trained using an 80-20 train-test split.
- **Results**: The model achieved a Mean Squared Error (MSE) of 115.41 and an R^2 score of 0.76, indicating a fairly strong predictive capability, meaning the features explain 76% of the variance in popularity.

2. Classification

```
features = ["energy", "loudness", "speechiness", "acousticness", "tempo","year"]
target = "explicit"
X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
```

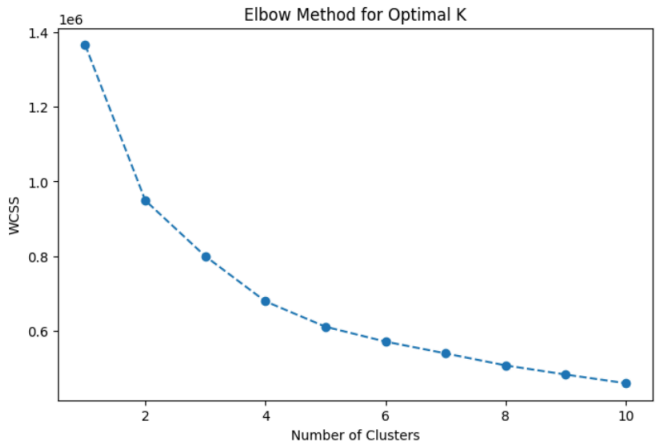
```
Accuracy: 0.95
Confusion Matrix:
[[30750   439]
 [ 1213  1729]]
```

- **Columns Used**: The classification model predicts explicitness of a song using energy, loudness, speechiness, acousticness, tempo, and year, as these features influence whether a song contains explicit content.
- **Approach**: A Random Forest Classifier was trained with 100 decision trees, using an 80-20 train-test split.

- **Results**: The model achieved a 95% accuracy, with the confusion matrix showing correct and incorrect classifications of explicit and non-explicit songs. The high accuracy suggests strong predictive performance.

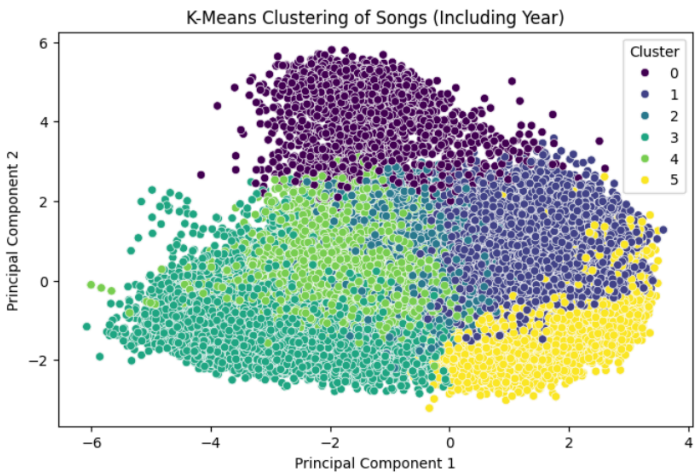
3. Clustering

```
features = ["danceability", "energy", "loudness", "speechiness", "acousticness", "instrumentalness", "tempo", "ye
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[features])
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("Elbow Method for Optimal K")
plt.show()
```




```
optimal_k = 6
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df["Cluster"] = kmeans.fit_predict(X_scaled)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
df["PCA1"] = X_pca[:, 0]
df["PCA2"] = X_pca[:, 1]
```

```
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df["PCA1"], y=df["PCA2"], hue=df["Cluster"], palette="viridis")
plt.title("K-Means Clustering of Songs (Including Year)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Cluster")
plt.show()
df[["year", "danceability", "energy", "loudness", "speechiness", "acousticness", "tempo", "Cluster"]].head()
```



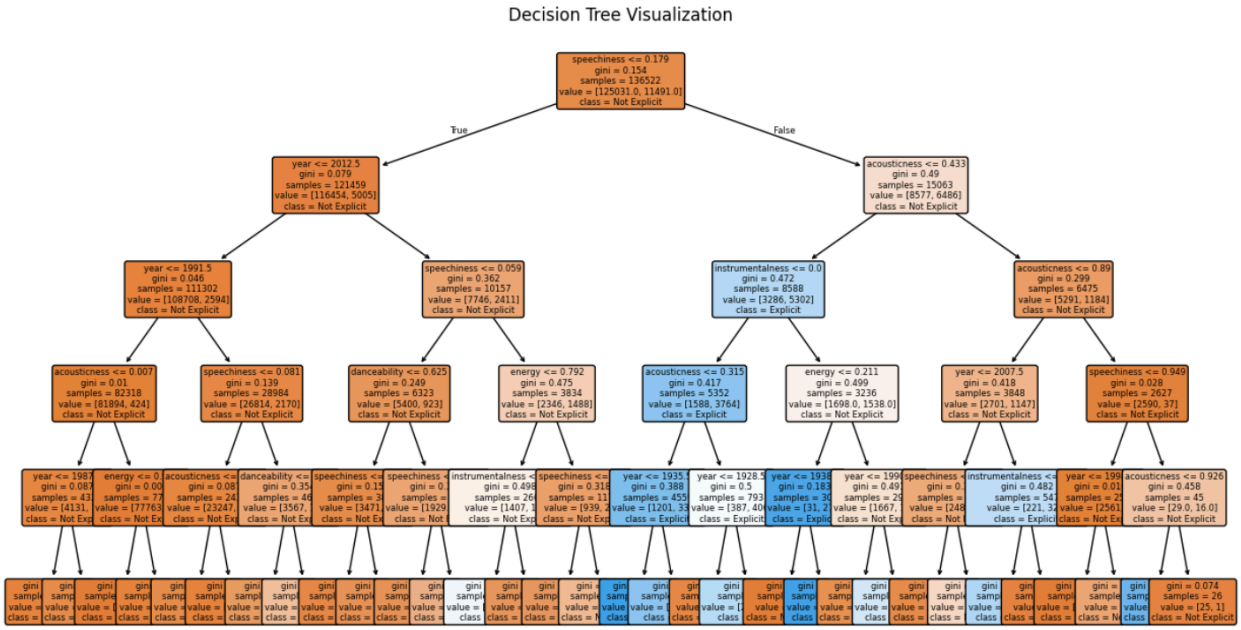
	year	danceability	energy	loudness	speechiness	acousticness	tempo	Cluster
0	1921	0.279	0.211	-20.096	0.0366	0.982	80.954	3
1	1921	0.819	0.341	-12.441	0.4150	0.732	60.936	4
2	1921	0.328	0.166	-14.850	0.0339	0.961	110.339	3
3	1921	0.275	0.309	-9.316	0.0354	0.967	100.109	4
4	1921	0.418	0.193	-10.096	0.0380	0.957	101.665	4

- **Columns Used**: Clustering was performed using danceability, energy, loudness, speechiness, acousticness, instrumentalness, tempo, and year, as these features help group songs with similar characteristics.
- **Approach**: K-Means clustering was applied after standardizing the data, with PCA used for visualization. The Elbow Method determined the optimal number of clusters (K=6).
- **Results**: Songs were grouped into six clusters, with trends emerging based on acoustic properties and time (year).

4. Decision Tree

```
\features = ["danceability", "energy", "loudness", "speechiness", "acousticness", "instrumentalness", "tempo", "year"]
X = df[features]
y = df["explicit"]  # Target variable (Binary: 0 = Not Explicit, 1 = Explicit)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)  # Limit depth to prevent overfitting
dt_model.fit(X_train, y_train)
y_pred = dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(15, 8))
plot_tree(dt_model, feature_names=features, class_names=["Not Explicit", "Explicit"], filled=True, rounded=True, fontsize=6)
plt.title("Decision Tree Visualization")
plt.show()
```

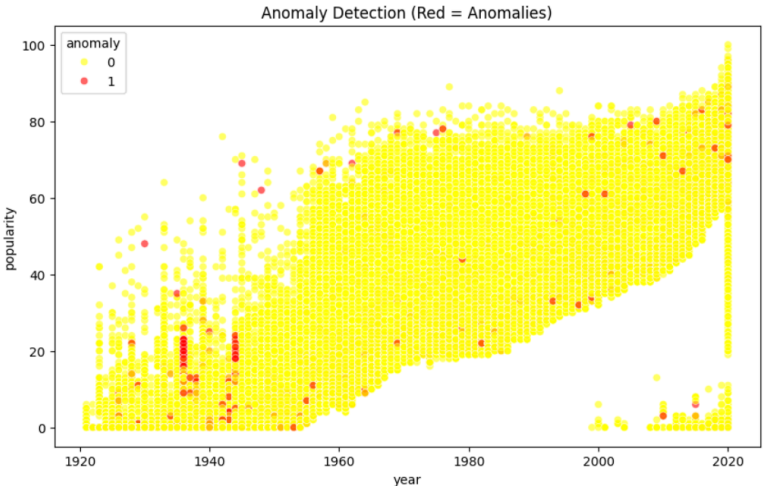
```
Accuracy: 0.94
Confusion Matrix:
[[30293   896]
 [ 1229  1713]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97     31189
           1       0.66      0.58      0.62      2942

    accuracy                           0.94     34131
   macro avg       0.81      0.78      0.79     34131
weighted avg       0.93      0.94      0.94     34131
```



- **Columns Used**: The decision tree classifier was built using danceability, energy, loudness, speechiness, acousticness, instrumentalness, tempo, and year to predict whether a song is explicit (1) or not explicit (0).
- **Approach**: A Decision Tree model (max depth=5) was trained on an 80-20 train-test split, balancing complexity and generalizability.
- **Results**: The model achieved 94% accuracy, with strong performance in detecting non-explicit songs but lower recall for explicit ones (58%).

5. Anomaly Detection

```
anomaly_features = df[['popularity', 'year', 'energy', 'liveness', 'speechiness', 'tempo']]
iso_forest = IsolationForest(n_estimators=100, contamination=0.02, random_state=42)
df['anomaly'] = iso_forest.fit_predict(anomaly_features)
df['anomaly'] = df['anomaly'].apply(lambda x: 1 if x == -1 else 0)
print("Total Anomalies Detected:", df['anomaly'].sum())
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['year'], y=df['popularity'], hue=df['anomaly'], palette={0: "yellow", 1: "red"}, alpha=0.6)
plt.title("Anomaly Detection (Red = Anomalies)")
plt.show()
```

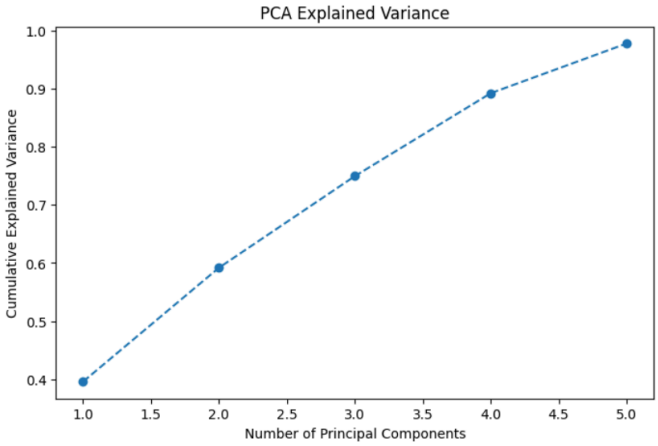



```
Total Anomalies Detected: 3414
```

- **Columns Used**: The Isolation Forest algorithm was applied using popularity, year, energy, liveness, speechiness, and tempo to detect anomalies in the dataset. These features were chosen as they significantly influence song characteristics and potential outliers.
- **Approach**: The model was trained with 100 estimators and a contamination rate of 2%, meaning it assumes about 2% of the data are anomalies. Anomalies were labeled as 1 (outlier) and 0 (normal).
- **Results**: A total of 3,414 anomalies were detected, and the scatter plot visualizes anomalies (red points) in the year vs. popularity space, highlighting unusual song behaviors.

6. Dimensionality Reduction

```
features = df[['popularity', 'year', 'energy', 'liveness', 'speechiness', 'tempo']]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
pca = PCA(n_components=0.95)  # Keeps enough PCs to explain 95% variance
pca_features = pca.fit_transform(scaled_features)
print(f"Number of Principal Components Retained: {pca.n_components_}")
pca_df = pd.DataFrame(pca_features, columns=[f'PC{i+1}' for i in range(pca.n_components_)])
df_pca = pd.concat([df, pca_df], axis=1)
plt.figure(figsize=(8, 5))
plt.plot(range(1, pca.n_components_+1), np.cumsum(pca.explained_variance_ratio_), marker='o', linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA Explained Variance')
plt.show()
```




```
Number of Principal Components Retained: 5
```

- Approach: The model retained 5 principal components, keeping enough features to explain 95% of the variance in the data while reducing dimensionality.
- Results: A variance plot was generated to visualize how much information each component retains, helping to balance dimensionality reduction and data preservation.

7. Ensemble Methods

Bagging Regression

```
bagging_reg = BaggingRegressor(RandomForestRegressor(n_estimators=100),
                               n_estimators=50, random_state=42)
bagging_reg.fit(X_train, y_train)
y_pred = bagging_reg.predict(X_test)

print(f"Bagging Regression MSE: {mean_squared_error(y_test, y_pred):.2f}")
print(f"Bagging Regression R²: {r2_score(y_test, y_pred):.2f}")
```

```
Bagging Regression MSE: 0.04
Bagging Regression R²: 0.54
```

- Columns: Used danceability, energy, loudness, speechiness, acousticness, instrumentalness, tempo, and year to predict popularity.
- Results: MSE = 0.04, R^2 = 0.54, meaning moderate prediction accuracy.

ADABoost Regression

```
ada_reg = AdaBoostRegressor(n_estimators=50, random_state=42)
ada_reg.fit(X_train, y_train)
y_pred = ada_reg.predict(X_test)

print(f"AdaBoost Regression MSE: {mean_squared_error(y_test, y_pred):.2f}")
print(f"AdaBoost Regression R²: {r2_score(y_test, y_pred):.2f}")
```

```
AdaBoost Regression MSE: 0.06
AdaBoost Regression R²: 0.26
```

- **Columns Used:** The regression model used highly correlated features like danceability, energy, loudness, speechiness, acousticness, instrumentalness, tempo, and year to predict popularity.
- **Approach:** AdaBoost was applied to improve performance by training weak learners sequentially, adjusting for errors at each step.
- **Results:** The model achieved a **Mean Squared Error (MSE) of 0.06** and an **R² score of 0.26**, indicating moderate predictive performance but room for improvement.

Stacking Regression

```python
stacking_reg = StackingRegressor(
    estimators=[('rf', RandomForestRegressor(n_estimators=100)),
                ('gb', GradientBoostingRegressor(n_estimators=100))],
    final_estimator=LinearRegression()
)
stacking_reg.fit(X_train, y_train)
y_pred = stacking_reg.predict(X_test)

print(f"Stacking Regression MSE: {mean_squared_error(y_test, y_pred):.2f}")
print(f"Stacking Regression R²: {r2_score(y_test, y_pred):.2f}")
```

```
Stacking Regression MSE: 0.04
Stacking Regression R²: 0.50
```

- Columns: Used danceability, energy, loudness, speechiness, acousticness, instrumentalness, tempo, and year to predict popularity.
- Results: MSE = 0.04, R² = 0.50, showing moderate predictive power.

Conclusion

- **Regression**: Linear Regression gave R² = 0.76, but Bagging Regression improved it to 0.54 with lower MSE.
- **Classification**: Random Forest had 95% accuracy in predicting explicit songs, outperforming Decision Trees (94%).
- **Clustering & Anomalies**: K-Means found 6 clusters, and Isolation Forest detected 3,414 anomalies in song data.
- **Dimensionality Reduction**: PCA retained 95% variance, reducing feature space.
- **Ensemble Methods**: Bagging and Stacking boosted predictive power, with Stacking achieving MSE = 0.04, R² = 0.50.