REGRESSION PROBLEM

CS985GROUP46

Aishwarya Kurhe - 202382549

Santosh Bangera - 202366764

Krish Dasani - 202388555

Ayesha Gaikwad - 202363500

Obaretin Abieyuwa - 202394985

Introduction

For the purpose of the given task, we are provided with a dataset from Spotify that comprises different attributes about songs, including a popularity score. The primary goal of the problem is to predict the popularity score of a song by using regression method in machine learning with Python. To ensure the best possible outcome, extensive data exploration, understanding, cleaning and preprocessing must be completed prior to model implementation. In order to address the problem, we have tried multiple models for interpreting the results and opted for the one that performed comparatively better than others.

[1]:
```
#importing libraries
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.linear_model import Ridge
```

Data Summary

The process is initialized by loading and reading the datasets into the memory. For understanding the structure of the loaded data, we used only top few rows that helped in displaying the contents in an organised manner.

[2]:
```
# Load the datasets
train_data = "CS98XRegressionTrain.csv"
test_data = "CS98XRegressionTest.csv"
train_data = pd.read_csv(train_data)
test_data = pd.read_csv(test_data)
train_data.head()
```

[2]:

	Id	title	artist	top genre	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop
0	1	My Happiness	Connie Francis	adult standards	1996	107	31	45	-8	13	28	150	75	3	44
1	2	Unchained Melody	The Teddy Bears	NaN	2011	114	44	53	-8	13	47	139	49	3	37
2	3	How Deep Is Your Love	Bee Gees	adult standards	1979	105	36	63	-9	13	67	245	11	3	77
3	4	Woman in Love	Barbra Streisand	adult standards	1980	170	28	47	-16	13	33	232	25	3	67
4	5	Goodbye Yellow Brick Road - Remastered 2014	Elton John	glam rock	1973	121	47	56	-8	15	40	193	45	3	63

Data cleaning and preprocessing

In data cleaning and preprocessing, we initially dropped the irrelevant columns from the dataset that were deemed to be unnecessary due to a weak correlation with the popularity score as shown in the heatmap. The columns dropped are ‘Id’, ‘title’, ‘artist’, ‘spch’, ‘bpm’, ‘val’, ‘live’, ‘year’, ‘dnce’. The next important step is to handle the missing values in the ‘top genre’ column, so we fill these with the mode. We then used LabelEncoder to convert ‘top genre’ column into string and combined it from both datasets before encoding to ensure consistency. Finally, we split the encoded ‘top genre’ back into respective datasets.

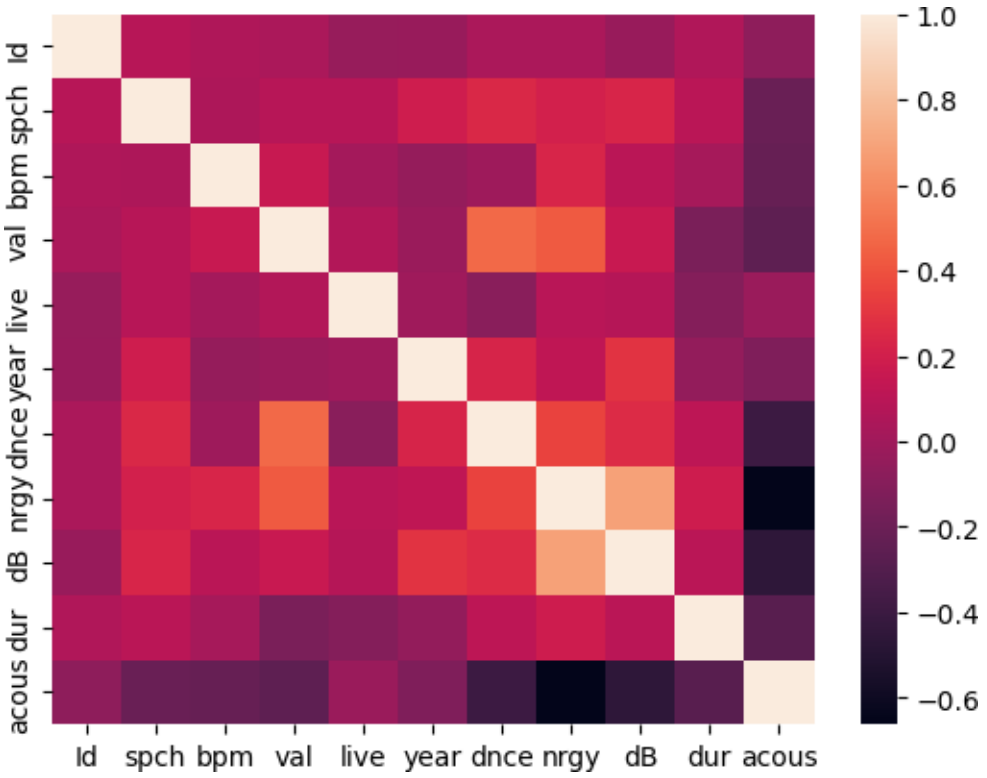
[3]:
```
# Data cleaning and preprocessing
# Remove 'Id', 'title', and 'artist' columns
train_data_cleaned = train_data.drop(['Id', 'title', 'artist'], axis=1)
test_data_cleaned = test_data.drop(['Id', 'title', 'artist'], axis=1)

# Handle missing values in 'top genre'
train_data_cleaned['top genre'].fillna(train_data_cleaned['top genre'].
 ↪mode()[0], inplace=True)
test_data_cleaned['top genre'].fillna(test_data_cleaned['top genre'].mode()[0],␣
 ↪inplace=True)

# Convert 'top genre' to string in both datasets
train_data_cleaned['top genre'] = train_data_cleaned['top genre'].astype(str)
test_data_cleaned['top genre'] = test_data_cleaned['top genre'].astype(str)

# Combine 'top genre' columns from both datasets before encoding
all_genres = pd.concat([train_data_cleaned['top genre'], test_data_cleaned['top␣
 ↪genre']], axis=0)

# Encoding 'top genre' column
encoder = LabelEncoder()
all_genres_encoded = encoder.fit_transform(all_genres)

# Splitting the encoded 'top genre' back into respective datasets
train_data_cleaned['top genre'] = all_genres_encoded[:len(train_data_cleaned)]
test_data_cleaned['top genre'] = all_genres_encoded[len(train_data_cleaned):]
train_data_cleaned.head()
```

[3]:

	top genre	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop
0	1	1996	107	31	45	-8	13	28	150	75	3	44
1	1	2011	114	44	53	-8	13	47	139	49	3	37
2	1	1979	105	36	63	-9	13	67	245	11	3	77
3	1	1980	170	28	47	-16	13	33	232	25	3	67
4	76	1973	121	47	56	-8	15	40	193	45	3	63

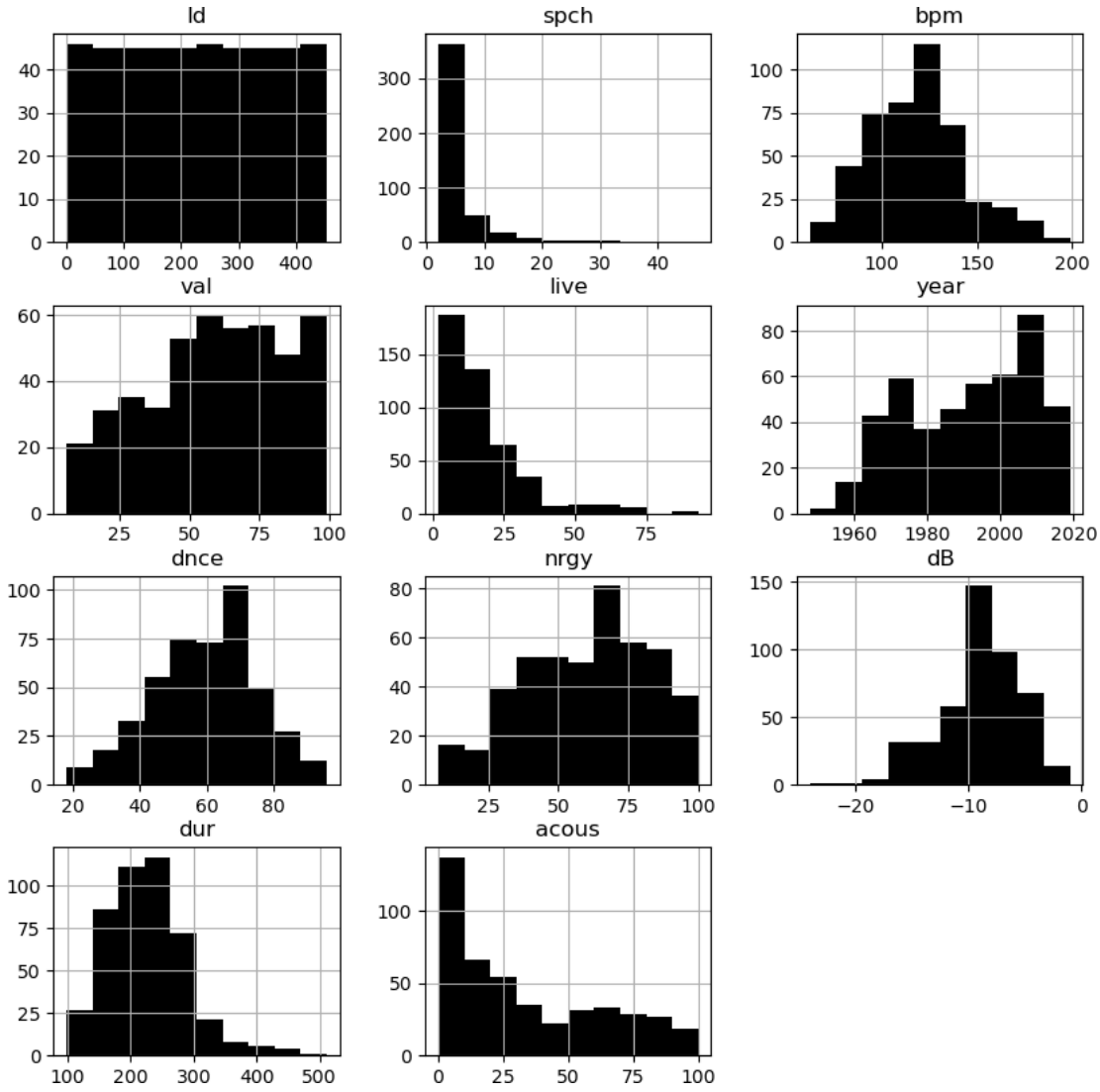
[4]:
```
df=␣
 ↪train_data[['Id','spch','bpm','val','live','year','dnce','nrgy','dB','dur','acous']]
corr = df.corr()
sns.heatmap(corr)
```

[4]: <Axes: >



The collection of histograms have provided a comprehensive visualization, each representing the distribution of various attributes in the dataset. To illustrate, the histogram for 'dnce' suggests that the data is somewhat normally distributed with a slight skew towards higher values, indicating a selection of music that tends to be more danceable. Similarly, for 'nrgy' it shows a fairly even distribution across the range with a slight concentration in the mid-range values. Moreover, in case of 'dB', the graph shows values ranging from -20 to 0 dB, with the data skewed towards the higher end, close to 0 dB, which might indicate a collection of relatively loud tracks.

[5]:
```
numerical_columns = list(df.dtypes[df.dtypes != 'object'].index.values)
df.loc[:,numerical_columns].hist(figsize=(10,10),color='black');
```



Model Training

To train the model, we split the training data into training and validation sets (80% train, 20% validation) after defining features (X_train) and target variable (Y_train), then the Root Mean Squared Error (RMSE) has been calculated. With the help of this training model, we predicted the popularity score of the test dataset.

[6]:
```
# Model training
X_train = train_data_cleaned.drop("pop", axis=1)
y_train = train_data_cleaned["pop"]

# Splitting the training data for validation (80% train, 20% validation)
```

```
X_train_split, X_val_split, y_train_split, y_val_split =␣
↪train_test_split(X_train, y_train, test_size=0.3, random_state=42)
```

Hyperparameter Tuning

Before implementing the regression models, the grid search technique was used to identify optimal hyperparameters. Hyperparameter tuning is a crucial step to follow that significantly impacted the overall performance. It helped to ensure that our regression model is well-suited to the characteristics of the data, providing better predictions and efficient resource utilization. Furthermore, it proved to be a key part of the iterative process in developing robust and effective machine learning algorithms.

[7]:
```
#parameter tuning for Random Forest
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid,␣
↪cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_split, y_train_split)
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
print("Best Parameters:", best_params)
y_val_pred = best_model.predict(X_val_split)
val_rmse = np.sqrt(mean_squared_error(y_val_split, y_val_pred))
print(f"Optimized Random Forest Validation RMSE: {val_rmse}")
```

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split':
2, 'n_estimators': 200}
Optimized Random Forest Validation RMSE: 11.185972247787795
```

Optimized Random Forest Validation RMSE: 11.185972247787795

RandomForestRegressor

The RandomForestRegressor model ultimately got picked for tackling the regression problem. There are several reasons for particularly choosing the method, one of them is its robust nature. Another most important reason is the high accuracy that it offered for the given data. The model provides a feature oriented as well as scalable approach to the dataset. Apart from this, it also reduces the issue of overfitting.

[8]:
```
# Initialize RandomForestRegressor
random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
random_forest_model.fit(X_train_split, y_train_split)
```

```
# Validate the model
y_val_pred = random_forest_model.predict(X_val_split)
val_rmse = np.sqrt(mean_squared_error(y_val_split, y_val_pred))
print(f"Random Forest Validation RMSE(Random Forest): {val_rmse}")

# Predicting the 'pop' scores on the testing data
test_predictions = random_forest_model.predict(test_data_cleaned)

# Adding the predictions to the test_data dataframe
test_data_with_predictions = test_data_cleaned.copy()
test_data_with_predictions['predicted_pop'] = test_predictions
submission = pd.DataFrame({
    'Id': test_data['Id'],
    'pop': test_predictions
})
submission_file_path = 'submission1.csv'
submission.to_csv(submission_file_path, index=False)
print(f"Submission file saved to: {submission_file_path}")
```

```
Random Forest Validation RMSE(Random Forest): 11.058737396019387
Submission file saved to: submission1.csv
```

Random Forest Validation RMSE(Random Forest): 11.058737396019387

Support Vector Regressor (SVR)

Support Vector Regressor (SVR) is one of the two other models that we tried, where we trained and validated the model before predicting the 'pop' scores on the testing data.

[9]:
```
#parameter tuning for SVR
param_grid_svr = {
    'C': [0.1, 1.0, 10.0, 100.0],  # Regularization parameter
    'epsilon': [0.01, 0.1, 0.2, 0.5],  # Epsilon in the epsilon-SVR model
    'gamma': ['scale', 'auto'],  # Kernel coefficient for 'rbf', 'poly', and␣
 ↪'sigmoid'
}
# Assuming X_train_split and y_train_split are your training data and labels
grid_search_svr = GridSearchCV(SVR(), param_grid_svr, cv=5,␣
 ↪scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_svr.fit(X_train_split, y_train_split)
best_params_svr = grid_search_svr.best_params_
best_model_svr = grid_search_svr.best_estimator_
print("Best Parameters (SVR):", best_params_svr)
y_val_pred_tuned = best_model_svr.predict(X_val_split)
val_rmse_tuned = np.sqrt(mean_squared_error(y_val_split, y_val_pred_tuned))
print(f"Validation RMSE (Tuned SVR): {val_rmse_tuned}")
```

```
Best Parameters (SVR): {'C': 100.0, 'epsilon': 0.5, 'gamma': 'scale'}
Validation RMSE (Tuned SVR): 12.352014145597666
```

Validation RMSE (Tuned SVR): 12.352014145597666

[10]:
```python
#svr model
svr_model = SVR(C=1.0, epsilon=0.2)

# Train the model
svr_model.fit(X_train_split, y_train_split)

# Validate the model
y_val_pred = svr_model.predict(X_val_split)
val_rmse = np.sqrt(mean_squared_error(y_val_split, y_val_pred))
print(f"Validation RMSE (SVR): {val_rmse}")

# Predicting the 'pop' scores on the testing data
test_predictions = svr_model.predict(test_data_cleaned)

# Creating submission DataFrame and saving to a CSV file
submission = pd.DataFrame({
    'Id': test_data['Id'],
    'pop': test_predictions
})
submission_file_path = 'submission_svr.csv'
submission.to_csv(submission_file_path, index=False)

print(f"Submission file (SVR) saved to: {submission_file_path}")
```

```
Validation RMSE (SVR): 14.515512613236899
Submission file (SVR) saved to: submission_svr.csv
```

Validation RMSE (SVR): 14.515512613236899

Linear Regression

Another one of the models that we decided to explore is linear regression, in which the 'pop' scores were predicted on the testing data in a similar way as SVR.

[11]:
```python
#parameter tuning for Linear Regression
param_grid_ridge = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100]
}
grid_search_ridge = GridSearchCV(Ridge(), param_grid_ridge, cv=5,␣
↪scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_ridge.fit(X_train_split, y_train_split)
best_params_ridge = grid_search_ridge.best_params_
best_model_ridge = grid_search_ridge.best_estimator_
```

```
print("Best Parameters (Ridge):", best_params_ridge)
y_val_pred_ridge = best_model_ridge.predict(X_val_split)
val_rmse_ridge = np.sqrt(mean_squared_error(y_val_split, y_val_pred_ridge))
print(f"Validation RMSE (Tuned Ridge): {val_rmse_ridge}")
```

```
Best Parameters (Ridge): {'alpha': 100}
Validation RMSE (Tuned Ridge): 11.971187315536783
```

Validation RMSE (Tuned Ridge): 11.971187315536783

[12]:
```
#linear regression model
linear_regression_model = LinearRegression()

# Train the model
linear_regression_model.fit(X_train_split, y_train_split)

# Validate the model
y_val_pred_lr = linear_regression_model.predict(X_val_split)
val_rmse_lr = np.sqrt(mean_squared_error(y_val_split, y_val_pred_lr))
print(f"Validation RMSE (Linear Regression): {val_rmse_lr}")

# Predicting the 'pop' scores on the testing data
test_predictions_lr = linear_regression_model.predict(test_data_cleaned)  #␣
↪Ensure to select features if not done globally

# Creating submission DataFrame and saving to a CSV file
submission_lr = pd.DataFrame({
    'Id': test_data['Id'],
    'pop': test_predictions_lr
})
submission_file_path_lr = 'submission_linear_regression.csv'
submission_lr.to_csv(submission_file_path_lr, index=False)

print(f"Submission file (Linear Regression) saved to:␣
↪{submission_file_path_lr}")
```

```
Validation RMSE (Linear Regression): 11.992724581303627
Submission file (Linear Regression) saved to: submission_linear_regression.csv
```

Validation RMSE (Linear Regression): 11.992724581303627

Conclusion

In our pursuit of predicting the popularity score of a song for the given task, we thoroughly cleaned and preprocessed the data, applied exploratory data analysis as well as tested multiple machine learning models such as RandomForestRegressor, Support Vector Regressor (SVR) and linear regression. Before training any of these models, however, we needed to use hyperparameter tuning for optimal performance. As we experimented with all these models, the RMSE results from

RandomForestRegressor model were the most promising and hence it became the final choice to achieve accurate solution for our problem.