INF1340 – Maher Elshakankiri

Danika Mariam, Chloe Li, Rosa Lee

Project Title:

Taylor Swift’s Spotify Songs – Descriptive, Predictive, Diagnostic Analysis

The Final Descriptive script outputs information about Taylor Swift's songs, providing insights into the characteristics of her music. The data file had a total of 528 rows and 17 columns. The script generates a histogram plot for song popularity using the Seaborn library. The below information was identified:

- Overview of non-null values in the dataset.
- List of Taylor Swift's albums.
- Average scores and standard deviation of numeric columns.
- Identification of songs with the maximum and minimum values for various attributes.
- Histogram plot of song popularity.

The Final Diagnostic script specifically explores the correlation between various song characteristics in Taylor Swift's dataset. We leverage correlation matrix and heatmap visualization to analyze the relationships among acousticness, danceability, energy, instrumentalness, liveness, loudness, speechiness, tempo, valence, duration, and popularity.

- Data loading and correlation matrix
- Heatmap visualization
- Scatterplots

The Final Predictive script conducts predictive analytics on Taylor Swift's songs using machine learning model and implements both Linear Regression and Logistic Regression models to predict the popularity of songs.

- Linear Regression:
    - Data Preparation
    - Model Training
    - Prediction and Evaluation
- Logistic Regression:
    - Data Preparation
    - Data Validation
    - Model Training and Evalution
    - Model Coefficients and Feature Importance
    - Probability Plots

Motivation:

The motivation for this data analysis is we are big Taylor Swift Fans and wanted to summarize the data as we are aware she has a lot of different types of songs that are popular.

Build Status:

There are currently no bugs for all three scripts.

Code Style:

We have used python with pandas incorporating functions, so please install pandas before running the file.

Running the Script:

- Open the script in a Google Colab environment.
- Run each code cell sequentially.

Screenshots:

Taylor Swift	FALSE	2006-10-24	14	A Perfectly Good He	Taylor Swift	NA	TRUE	NA	NA	2008-03-18	0.483	0.751
Taylor Swift	FALSE	2006-10-24	15	Teardrops On My Gu	Taylor Swift	NA	TRUE	NA	NA	2008-03-18	0.459	0.753
Fearless	FALSE	2008-11-11	1	Fearless	Taylor Swift	NA	FALSE	2008-10-14	2010-01-03	2008-10-14	0.598	0.714
Fearless	FALSE	2008-11-11	2	Fifteen	Taylor Swift	NA	FALSE	NA	2009-08-30	2008-11-11	0.559	0.636

Importing the data into panda data frame

```
import pandas as pd
```

The following code imports the csv file from Google

```
# import the drive
from google.colab import drive
drive.mount("/drive", force_remount=True)
```

Descriptive stats:

Importing panda matplot.lib.pyplot, and seaborn packages.

```
# import packages
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.backends.backend_pdf import PdfPages
```

Identifying the number of non-null values that are in the data.

```
# let's look at how many non-null values are in the dataset
songs.info()
```

```
Mounted at /drive
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 530 entries, 0 to 529
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        530 non-null    int64
 1   name              530 non-null    object
 2   album             530 non-null    object
 3   release_date      530 non-null    object
 4   track_number      530 non-null    int64
 5   id                530 non-null    object
 6   uri               530 non-null    object
 7   acousticness      530 non-null    float64
 8   danceability      530 non-null    float64
 9   energy            530 non-null    float64
 10  instrumentalness  530 non-null    float64
 11  liveness          530 non-null    float64
 12  loudness          530 non-null    float64
 13  speechiness       530 non-null    float64
 14  tempo             530 non-null    float64
 15  valence           530 non-null    float64
 16  popularity        530 non-null    int64
 17  duration_ms       530 non-null    int64
dtypes: float64(9), int64(4), object(5)
memory usage: 74.7+ KB
```

The print_album_names() function used to print the list of Taylor Swift album names.

```
# list out album names
def print_album_names():
  album_names = songs["album"].unique()
  num_albums = len(album_names)
  print('Taylor Swift has released', num_albums,  'albums: \n')
  for album in album_names:
    print(album)
    print()
```

The print_avg_scores() and print_std() functions are used to calculate the average scores of the numerical values for danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, temp, time_signature, and duration_ms.

```
# some important average scores, (danceability, energy, loudness, speechiness,
# acousticness, instrumentalness, liveness, valence, temp, time_signature, duration_ms)
def print_avg_scores():
  average_scores = songs.mean(numeric_only=True)
  print("Mean of Numeric Columns:")
  print(average_scores[2:].to_frame())
  print()

#printing the sd of the code
def print_std():
  std_deviation = songs.std(numeric_only=True)
  print("Standard Deviation of Numeric Columns:")
  print(std_deviation[2:].to_frame())
  print()
```

The print_max_min() function calculates the max and min function of each variable, the below is an example of the calculation of the max and min of the variable danceability.

```
# report the max and min of each numeric score
def print_max_min():
  max_danceability = songs.loc[songs["danceability"] == songs['danceability'].max(), "name"].to_list()
  max_danceability = list(set(max_danceability))
  print("Taylor's most danceable song(s):", max_danceability)
  min_danceability = songs.loc[songs["danceability"] == songs['danceability'].min(), "name"].to_list()
  min_danceability = list(set(min_danceability))
  print("Taylor's least danceable song(s):", min_danceability)
  print()
```

Function hist_pop() to create the histogram for popularity.

```
# make histogram for popularity
def hist_pop():
  plt.figure()
  sns.histplot(songs["popularity"],bins='auto')
  plt.show()
```

More functions to create histograms for variables.

```
# make histogram for danceability
def hist_dance():
  plt.figure()
  sns.histplot(songs["danceability"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()

# make histogram for energy
def hist_energy():
  plt.figure()
  sns.histplot(songs["energy"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()

# make histogram for loudness
def hist_loud():
  plt.figure()
  sns.histplot(songs["loudness"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()
```

```python
# make histogram for loudness
def hist_loud():
  plt.figure()
  sns.histplot(songs["loudness"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()

# make histogram for speechiness
def hist_speech():
  plt.figure()
  sns.histplot(songs["speechiness"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()

# make histogram for acousticness
def hist_acoustic():
  plt.figure()
  sns.histplot(songs["acousticness"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()

# make histogram for liveness
def hist_liveness():
  plt.figure()
  sns.histplot(songs["liveness"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()
```

```python
# make histogram for valence
def hist_valence():
  plt.figure()
  sns.histplot(songs["valence"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()

# make histogram for tempo
def hist_tempo():
  plt.figure()
  sns.histplot(songs["tempo"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()

# make histogram for duration
def hist_duration():
  plt.figure()
  sns.histplot(songs["duration_ms"],bins='auto')
  plt.savefig(output_plots, format='pdf')
  plt.show()
```

Main() function to print all of the previous functions we have defined.

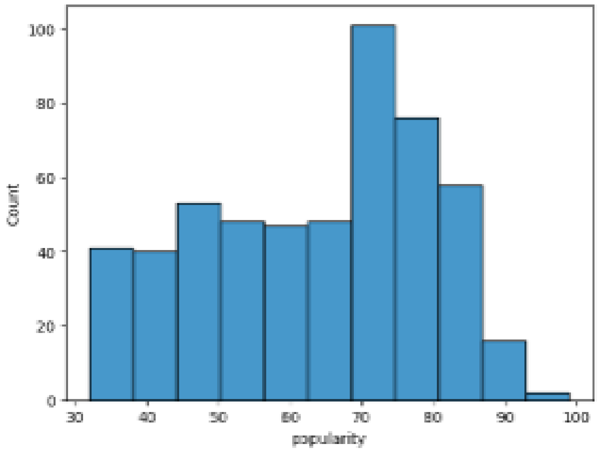
```python
# main program
def main():
  print_album_names()
  print_avg_scores()
  print_std()
  print_max_min()
  hist_pop()

main()
```

Final print:

```
Taylor Swift has released 27 albums:

1989 (Taylor's Version) [Deluxe]

1989 (Taylor's Version)

Speak Now (Taylor's Version)

Midnights (The Til Dawn Edition)

Midnights (3am Edition)

Midnights

Red (Taylor's Version)

Fearless (Taylor's Version)

evermore (deluxe version)

evermore

folklore: the long pond studio sessions (from the Disney+ special) [deluxe edition]

folklore (deluxe version)

folklore

Lover

reputation

reputation Stadium Tour Surprise Song Playlist

1989 (Deluxe Edition)

1989

Red (Deluxe Edition)

Red

Speak Now World Tour Live

Speak Now (Deluxe Edition)

Speak Now

Fearless Platinum Edition

Fearless

Live From Clear Channel Stripped 2008

Taylor Swift

Mean of Numeric Columns:
```

```
Mean of Numeric Columns:
                              0
acousticness           0.319247
danceability           0.585285
energy                 0.574609
instrumentalness       0.004005
liveness               0.163492
loudness              -7.505434
speechiness            0.055889
tempo                122.332311
valence                0.397379
popularity            63.615094
duration_ms       239978.624528

Standard Deviation of Numeric Columns:
                             0
acousticness          0.327043
danceability          0.113121
energy                0.191565
instrumentalness      0.033163
liveness              0.142263
loudness              2.939040
speechiness           0.070300
tempo                30.000272
valence               0.199589
popularity           15.590790
duration_ms       46119.983031

Taylor's most danceable song(s): ['I Think He Knows']
Taylor's least danceable song(s): ['Change - Live From Clear Channel Stripped 2008']

Taylor's most energetic song(s): ['Haunted']
Taylor's least energetic song(s): ['State Of Grace - Acoustic']

Taylor's most loud song(s): ["Haunted (Taylor's Version)"]
Taylor's least loud song(s): ['I Know Places - Voice Memo']

Taylor's most wordy song(s): ['I Wish You Would - Voice Memo']
Taylor's least wordy song(s): ['Teardrops On My Guitar - Radio Single Remix']

Taylor's most acoustic song(s): ['It's Nice To Have A Friend']
Taylor's least acoustic song(s): ['State Of Grace']

Taylor's most instrumental song(s): ['Labyrinth']
Taylor's least instrumental song(s): ['my tears ricochet', 'Nothing New (feat. Phoebe Bridgers) (Taylor's Version) (From The Vault)', 'You All Over Me (feat. Maren Morris) (Taylor's Version) (From The Vault)', 'A Place in this World', 'Ours - Live/2011', "Forever Winter (Taylor's Version) (From The Vault)", 'Ours', 'ME! (f

Taylor's most live song(s): ['Better Than Revenge - Live/2011']
Taylor's least live song(s): ['I Knew You Were Trouble.', 'The Story Of Us']

Taylor's happiest sounding song(s): ['Shake It Off']
Taylor's saddest sounding song(s): ['Maroon']

Taylor's most popular song(s): ['Cruel Summer']
Taylor's least popular song(s): ['Jump Then Fall', 'Hey Stephen']

Taylor's fastest song(s): ["State Of Grace (Acoustic Version) (Taylor's Version)"]
Taylor's slowest song(s): ['this is me trying - the long pond studio sessions']
```

```
Taylor's longest song(s): ["All Too Well (10 Minute Version) (Taylor's Version) (From The Vault)"]
Taylor's shortest song(s): ['I Want You Back - Live/2011']
```



<u>Predictive stat</u>s:

Packages pandas, matplotlib, pyplot, seaborn, and numpy are downloaded for this script

```
# import packages
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from matplotlib.backends.backend_pdf import PdfPages

from sklearn.model_selection import train_test_split, learning_curve, KFold
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, make_scorer, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
```

Function lin_regression() is defined to plot the observed vs predicted data, providing a line of best fit, mean squared error, and r2 score.

```
def lin_regression():
  # Selecting features and target variable
  features = songs[['acousticness', 'danceability', 'energy', 'instrumentalness',
                  'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'duration_ms']]
  target = songs['popularity']

  # Scaling features
  scaler = StandardScaler()
  features_scaled = scaler.fit_transform(features)

  # Splitting the dataset into training and testing sets
  X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

  # Creating and training the linear regression model
  model = LinearRegression()
  model.fit(X_train, y_train)

  # Making predictions on the test set
  predictions = model.predict(X_test)
  mse = mean_squared_error(y_test, predictions)
  r2 = r2_score(y_test, predictions)

  # calculating line of best fit
  a, b = np.polyfit(y_test, predictions, 1)
  plt.plot(y_test, a*y_test + b, color='red', label='Best Fit Line')

  # Plotting the observed vs predicted values for visual comparison
  plt.scatter(y_test, predictions)
  plt.xlabel('Observed')
  plt.ylabel('Predicted')
  plt.title('Observed vs Predicted Popularity')
  plt.show()

  print('Mean Squared Error:', mse)
  print('R^2 Score:', r2)
```

The log_regression() function performs Logistic Regression with K-Fold cross-validation finding the best hyperparameters for predicting song popularity based on selected features. Providing insights into the optimal model configuration and its performance metrics.

```python
# Logistic Regression

def log_regression():
  global features
  features = songs[['acousticness', 'danceability', 'energy', 'instrumentalness',
                    'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'duration_ms']]
  target = songs['popularity']
  target = (target > target.median()).astype(int)

  scaler = StandardScaler()
  kfold = KFold(n_splits=10, random_state=42, shuffle=True)

  # Initialize the best model and parameters
  global model
  best_model = None
  best_params = {}
  best_accuracy = 0
  best_std = 0

  global X_train, X_test, y_train, y_test
  X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3, random_state=42)
  global X_train_scaled, X_test_scaled
  X_train_scaled = scaler.fit_transform(X_train)
  X_test_scaled = scaler.transform(X_test)
```

```python
for C in [0.001, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100]:
    for solver in ['newton-cg', 'lbfgs', 'liblinear', 'sag']:
        # Set up the model with the current set of parameters
        model = LogisticRegression(C=C, solver=solver, max_iter=10000)

        # List to store accuracy for each fold
        accuracy = []

        # Perform K-Fold cross-validation
        for train_idx, test_idx in kfold.split(features):
            # Split the data
            X_train, X_test = features.values[train_idx], features.values[test_idx]
            y_train, y_test = target.values[train_idx], target.values[test_idx]

            # Scale the features
            X_train = scaler.fit_transform(X_train)
            X_test = scaler.transform(X_test)

            # Train the model
            model.fit(X_train, y_train)

            # Calculate accuracy
            score = model.score(X_test, y_test)
            accuracy.append(score)
```

```python
        # Calculate the average accuracy and standard deviation
        avg_accuracy = np.mean(accuracy)
        std_accuracy = np.std(accuracy)

        # Update the best model if the current model is better
        if avg_accuracy > best_accuracy:
            best_model = model
            best_params = {'C': C, 'solver': solver}
            best_accuracy = avg_accuracy
            best_std = std_accuracy

  print("Best Model Parameters:", best_params)
  print("Best Cross-Validation Accuracy: "+ str(round(best_accuracy,2) * 100) + "%")
  print("Standard Deviation of CV Accuracy: "+ str(round(best_std,2) * 100)+ "%")
```

The coeff() function prints the coefficients and features importance.

```
def coeff():
  coefficients = model.coef_[0]
  # Get feature names
  feature_names = features.columns

  # Create a DataFrame to display coefficients and feature importance
  coefficients_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})

  # print coefficients
  print("Coefficients:")
  print(coefficients_df)

  # print 'feature_importance'
  print("\nFeature Importance:")
  print(coefficients_df[['Feature', 'Coefficient']].sort_values(by='Coefficient', key=abs, ascending=False))
```

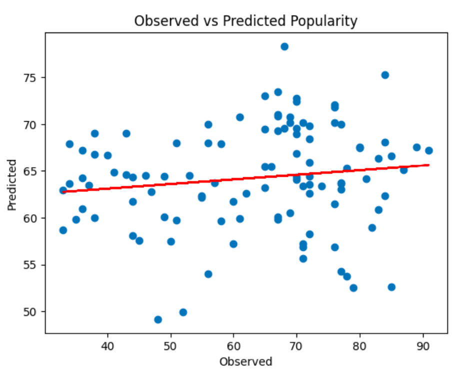
The log_plot() function plots the probability of the positive class against the actual values.

```
# Plot
def log_plot():
  y_probs = model.predict_proba(X_test)[:, 1]  # Probability of the positive class

  # Plotting the predicted probabilities against the actual values
  plt.figure(figsize=(10, 6))
  plt.scatter(range(len(y_test)), y_probs, c='r', label='Actual')
  plt.scatter(range(len(y_test)), y_test, alpha=0.5, edgecolor='k', label='Predicted')
  plt.title('Predicted Probabilities and Actual Values')
  plt.xlabel('Samples')
  plt.ylabel('Probability')
  plt.legend()
  plt.show()
```

Printing the functions using main().

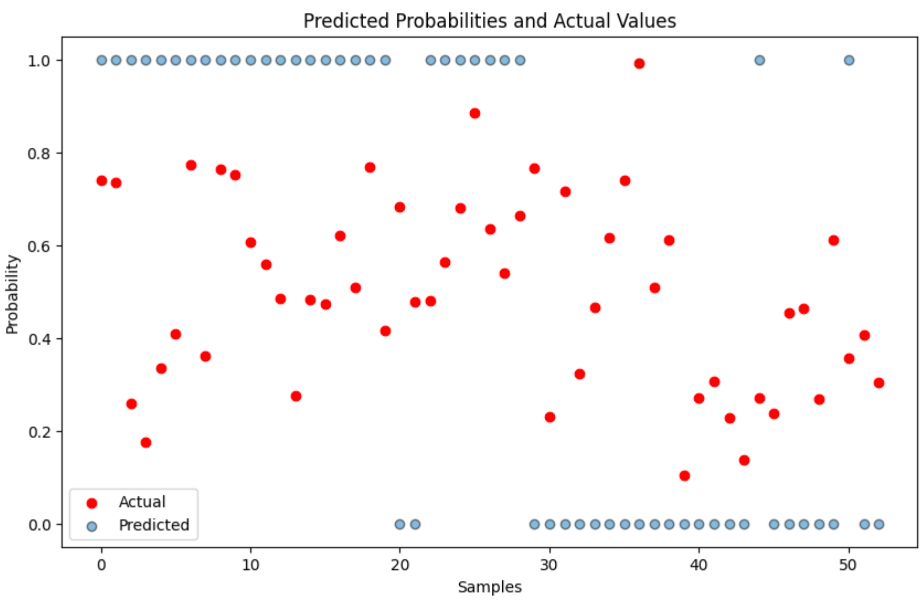
```
def main():
  lin_regression()
  log_regression()
  coeff()
  log_plot()

main()
```

The following was printed.




```
Mean Squared Error: 244.12007320828295
R^2 Score: -0.035057783355679106
Best Model Parameters: {'C': 0.05, 'solver': 'newton-cg'}
Best Cross-Validation Accuracy: 62.0%
Standard Deviation of CV Accuracy: 5.0%
```

```
Coefficients:
             Feature  Coefficient
0       acousticness    -0.303920
1       danceability     0.139714
2             energy     0.528223
3   instrumentalness     0.332272
4           liveness    -0.478203
5           loudness    -0.998470
6        speechiness     0.135944
7              tempo     0.057267
8            valence    -0.246278
9        duration_ms    -0.183269

Feature Importance:
             Feature  Coefficient
5           loudness    -0.998470
2             energy     0.528223
4           liveness    -0.478203
3   instrumentalness     0.332272
0       acousticness    -0.303920
8            valence    -0.246278
9        duration_ms    -0.183269
1       danceability     0.139714
6        speechiness     0.135944
7              tempo     0.057267
```



Diagnostic Stats:

Importing the pandas, matplotlib.pyplot, seaborn, and numpy files.

```python
# import packages
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.model_selection import train_test_split, learning_curve, KFold
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, make_scorer, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
```

Function corr() calculates the correlation and plots the heatmap

```python
def corr():
  # Selecting columns to find correlation
  columns = songs[['acousticness', 'danceability', 'energy', 'instrumentalness',
                  'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'duration_ms', 'popularity']]

  # Create the correlation matrix
  corr = columns.corr()

  # Set up the matplotlib figure
  f, ax = plt.subplots(figsize=(11, 9))

  # Set heatmap colors
  cmap = sns.color_palette("rocket", as_cmap=True)

  # Draw the heatmap with the mask and correct aspect ratio
  sns.heatmap(corr, cmap=cmap, vmax=.3, annot=True, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})

  plt.show()
```

Function t_test_acoustic() is a t-test to compare the compare 'acousticness' between songs with high and low popularity. The songs are categorized into high and low popularity based on the median. The function calculates the t-statistic and p-value.

```python
# Perform a t-test to compare the compare 'acousticness' between songs with high and low popularity

from scipy import stats

# Categorize songs into high and low popularity based the median

def t_test_acoustic():
  median_popularity = songs['popularity'].median()
  high_popularity = songs[songs['popularity'] >= median_popularity]
  low_popularity = songs[songs['popularity'] < median_popularity]

  # Perform the t-test
  t_stat, p_val = stats.ttest_ind(high_popularity['acousticness'], low_popularity['acousticness'])

  print("T-Statistic:", t_stat)
  print("P-Value:", p_val)

  # When a p-value less than 0.05, it is considered statistically significant
  if p_val < 0.05:
      print("The difference in acousticness between high and low popularity songs is statistically significant.")
  else:
      print("There're no significant difference in acousticness between high and low popularity songs.")
```

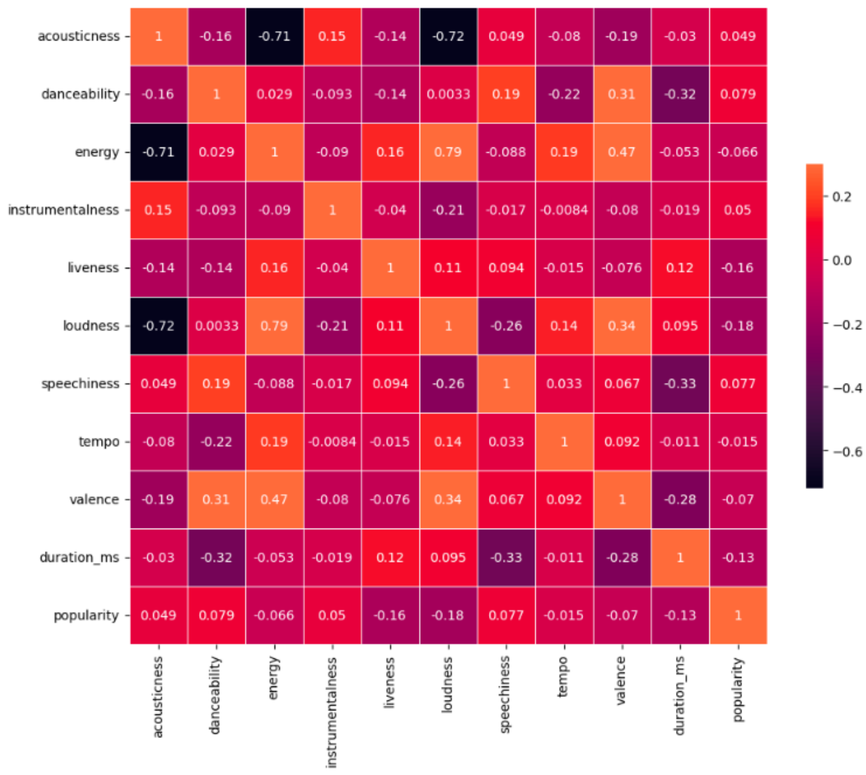
Function scatter(), facilitates the visual exploration of the relationship between each independent variable and the 'popularity' target variable through a series of scatterplots, aiding in the identification of potential trends or patterns in the data.

```python
def scatter():
  # Scatterplots between each independent variable and popularity
  # Create list for independent variables
  independent_variables = ['acousticness', 'danceability', 'energy', 'instrumentalness',
                        'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'duration_ms']

  popularity = 'popularity'
  n_rows = len(independent_variables) // 2 + len(independent_variables) % 2
  fig, axes = plt.subplots(nrows=n_rows, ncols=2, figsize=(14, n_rows * 4))

  # Flatten the axes array for easy indexing
  axes = axes.flatten()

  # Plot each independent variable vs the target variable
  for i, var in enumerate(independent_variables):
      sns.scatterplot(x=songs[var], y=songs[popularity], ax=axes[i])
      axes[i].set_xlabel(var)
      axes[i].set_ylabel(popularity)
      axes[i].set_title(f'Scatter plot of {var} vs {popularity}')

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

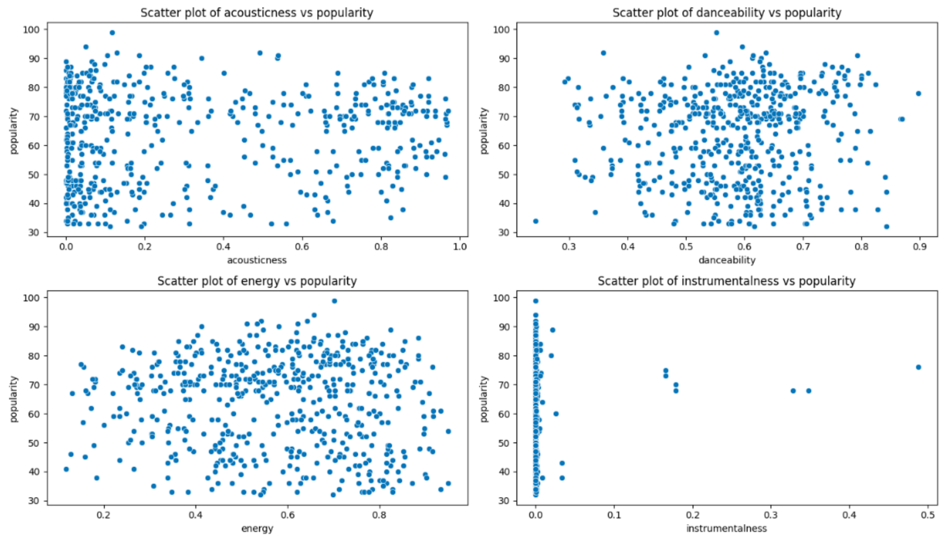
main() prints the final results.

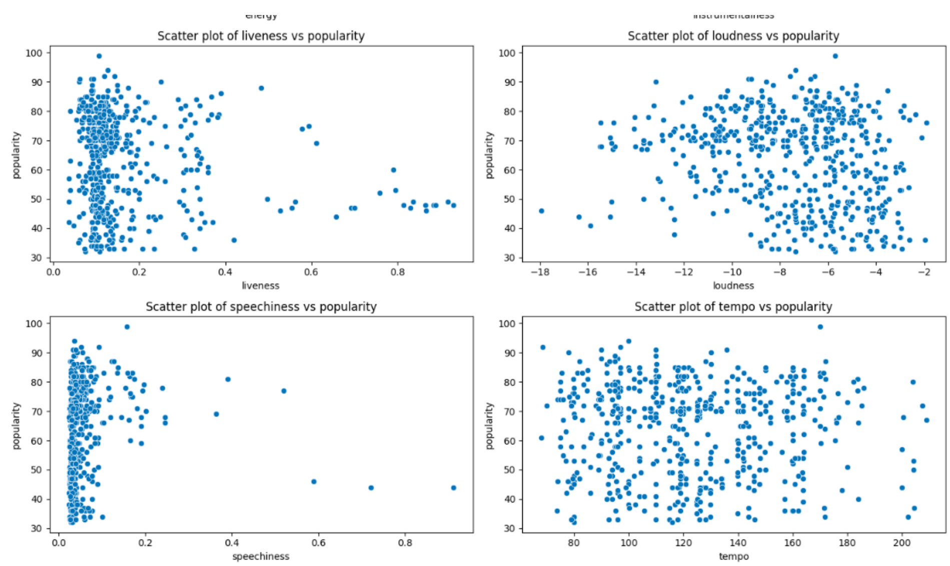
```python
# main program
def main():
  corr()
  t_test_acoustic()
  scatter()

main()
```

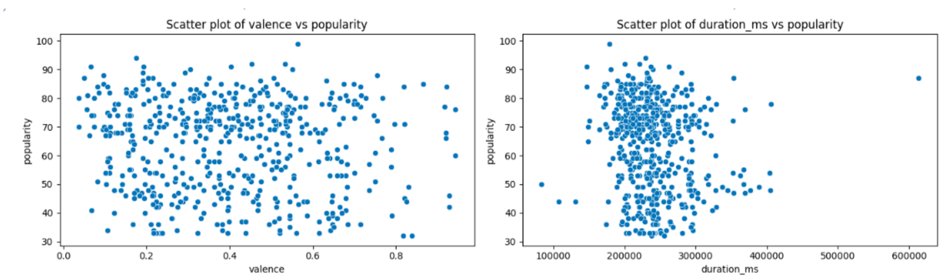
The final results are:



```
T-Statistic: 1.50717336016141
P-Value: 0.13236427957903582
There're no significant difference in acousticness between high and low popularity songs.
```







<u>Features:</u>

The code uses distinct panda functions to analyze the data.