```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import layers
import matplotlib.pyplot as plt
%matplotlib inline
```

```
df = pd.read_csv('sample_data/Animation.csv')
```

```
df.head()
```

	animeID	name	source	producers	genre	studio	episodes	airing	rank
0	1	Cowboy Bebop	Original	Bandai Visual	Action	Sunrise	26.0	False	26
1	5	Cowboy Bebop: Tengoku no Tobira	Original	Sunrise	Action	Bones	1.0	False	164
2	6	Trigun	Manga	Victor Entertainment	Action	Madhouse	26.0	False	255
3	7	Witch Hunter Robin	Original	Bandai Visual	Action	Sunrise	26.0	False	2371
4	8	Bouken Ou Beet	Manga	TV Tokyo	Adventure	Toei Animation	52.0	False	3544

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13631 entries, 0 to 13630
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   animeID          13631 non-null  int64  
 1   name             13631 non-null  object 
 2   source           13631 non-null  object 
 3   producers        7414 non-null   object 
 4   genre            13569 non-null  object 
 5   studio           8369 non-null   object 
 6   episodes         13349 non-null  float64
 7   airing           13631 non-null  bool   
 8   rank             13631 non-null  int64  
 9   popularity       13631 non-null  int64  
 10  members          13631 non-null  int64  
 11  favorites        13631 non-null  int64  
 12  reviewers        13631 non-null  int64  
 13  Animation_score  13518 non-null  float64
dtypes: bool(1), float64(2), int64(6), object(5)
memory usage: 1.4+ MB
```

```
# Using classification to determain genre for null values

        # PREPROCCESING
columsPredictGenre = ['source','producers','studio','episodes','rank','popularity','members','favorites','reviewers','Animation_score']
columsPredictScore = ['source','genre','producers','studio','episodes','rank','popularity','members','favorites','reviewers']
# Since Score has a little outlier and those who don'nt have score also don't have genre,
# This won't effect to predict genre
dfG = df[df.Animation_score.notna() == True]
yG = dfG['genre'].fillna('other genre')

dfS = dfG[columsPredictScore]
nullScores = df[df.Animation_score.notna() == False][columsPredictScore]
# y That i want to predict is df[df.Animation_score.notna() == False]
yS = dfG['Animation_score']


dfG = dfG[columsPredictGenre]
categorical_columns = dfG.select_dtypes(include=['object']).columns

numerical_columns = dfG.select_dtypes(include=['number']).columns
```

```
dfG[categorical_columns] = dfG[categorical_columns].fillna('Other')

# Fill null values in numerical columns with 0
dfG[numerical_columns] = dfG[numerical_columns].fillna(0)

  # For dfS
categorical_columns2 = dfS.select_dtypes(include=['object']).columns

numerical_columns2 = dfS.select_dtypes(include=['number']).columns

dfS[categorical_columns2] = dfS[categorical_columns2].fillna('Other')

# Fill null values in numerical columns with 0
dfS[numerical_columns2] = dfS[numerical_columns2].fillna(0)


# Check for null values
yG.isnull().any().any()
```

```
False
```

```
#dfg_encoded = pd.get_dummies(dfG, columns=['source','producers','studio'])

    # Select top categories to reduce the hot code encoding

def one_hot_encode_top(dfG, column_name,numb):
    top_categories = dfG[column_name].value_counts().index[:numb]
    dfG[column_name] = dfG[column_name].apply(lambda x: x if x in top_categories else 'Other')
    df_encoded = pd.get_dummies(dfG, columns=[column_name])
    return df_encoded

dfG = one_hot_encode_top(dfG, 'source',4)
dfG = one_hot_encode_top(dfG, 'producers',12)
dfG = one_hot_encode_top(dfG, 'studio',12)
dfG
```

	episodes	rank	popularity	members	favorites	reviewers	Animation_score	source_Game	source_Manga	source_Original	...
0	26.0	26	39	795733	43460	405664	8.81	0	0	1	...
1	1.0	164	449	197791	776	120243	8.41	0	0	1	...
2	26.0	255	146	408548	10432	212537	8.30	0	1	0	...
3	26.0	2371	1171	79397	537	32837	7.33	0	0	1	...
4	52.0	3544	3704	11708	14	4894	7.03	0	1	0	...
...	...	...	...	...	...	...	...	...	...	...	...
13553	1.0	10776	15213	38	0	21	2.57	0	0	1	...
13554	1.0	11138	15293	29	0	17	4.35	0	0	1	...
13556	1.0	13398	15346	19	0	7	4.57	0	0	1	...
13590	40.0	11345	15227	36	0	2	5.50	0	0	1	...
13625	12.0	13640	15271	45	0	6	6.17	0	0	0	...

13518 rows × 36 columns

```
# Modling for predicting genre using random forest
X_train, X_test, y_train, y_test = train_test_split(dfG, yG, test_size=0.2, random_state=42)

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=400, random_state=42)

# Train
rf_classifier.fit(X_train, y_train)

# Predict
predictions = rf_classifier.predict(X_test)

# Evaluate
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)
```

```
print(f'Accuracy: {accuracy}')
print(report)
```

```
Accuracy: 0.3768491124260355
               precision    recall  f1-score   support

       Action       0.40      0.65      0.49       631
    Adventure       0.35      0.25      0.30       259
         Cars       0.25      0.11      0.15         9
       Comedy       0.34      0.56      0.43       532
     Dementia       0.37      0.39      0.38        51
       Demons       0.00      0.00      0.00         7
        Drama       0.42      0.19      0.27       154
        Ecchi       0.00      0.00      0.00        16
      Fantasy       0.15      0.05      0.08        99
         Game       0.50      0.04      0.07        26
        Harem       0.00      0.00      0.00        14
   Historical       0.07      0.02      0.03        53
       Horror       0.00      0.00      0.00        10
         Kids       0.43      0.40      0.42       166
        Magic       0.00      0.00      0.00        15
 Martial Arts       0.00      0.00      0.00         1
        Mecha       1.00      0.10      0.17        21
     Military       0.00      0.00      0.00        15
        Music       0.54      0.48      0.51       213
      Mystery       0.00      0.00      0.00        31
       Parody       0.00      0.00      0.00        12
       Police       0.00      0.00      0.00         3
Psychological       0.00      0.00      0.00         8
      Romance       0.00      0.00      0.00        30
      Samurai       0.00      0.00      0.00         3
       School       0.00      0.00      0.00         7
       Sci-Fi       0.50      0.09      0.15        91
       Seinen       0.00      0.00      0.00         3
      Shounen       0.00      0.00      0.00         6
Slice of Life       0.14      0.04      0.06       151
        Space       0.00      0.00      0.00        12
       Sports       0.38      0.09      0.15        32
  Super Power       0.00      0.00      0.00         1
 Supernatural       0.00      0.00      0.00         6
     Thriller       0.00      0.00      0.00         1
  other genre       0.00      0.00      0.00        15

     accuracy                           0.38      2704
    macro avg       0.16      0.10      0.10      2704
 weighted avg       0.34      0.38      0.33      2704

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are 
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are 
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are 
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# Since the accuracy was low I tried a simple neural network

X = dfG.to_numpy()
y = yG.to_numpy()

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

numberOfGenres = df.genre.nunique()

# Define a neural network model
modelG = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(38, activation='relu'),
    layers.Dense(len(label_encoder.classes_), activation='softmax')  # Output layer with the number of classes
])

# Compile the model
modelG.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
modelG.fit(X_train, y_train, epochs=20, batch_size=64, validation_split=0.2)

# Evaluate the model on the test data
loss, accuracy = modelG.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```

```
Epoch 1/20
136/136 [==============================] - 4s 14ms/step - loss: 271.3604 - accuracy: 0.1200 - val_loss: 48.1036 - val_accuracy: 0.1:
Epoch 2/20
136/136 [==============================] - 0s 3ms/step - loss: 36.6695 - accuracy: 0.1134 - val_loss: 18.5277 - val_accuracy: 0.102(
Epoch 3/20
136/136 [==============================] - 0s 3ms/step - loss: 6.4399 - accuracy: 0.1881 - val_loss: 3.0868 - val_accuracy: 0.2080
Epoch 4/20
136/136 [==============================] - 0s 3ms/step - loss: 2.9775 - accuracy: 0.2085 - val_loss: 2.8636 - val_accuracy: 0.2048
Epoch 5/20
136/136 [==============================] - 0s 3ms/step - loss: 2.7821 - accuracy: 0.2081 - val_loss: 2.7125 - val_accuracy: 0.2043
Epoch 6/20
136/136 [==============================] - 0s 4ms/step - loss: 2.6652 - accuracy: 0.2075 - val_loss: 2.6231 - val_accuracy: 0.2043
Epoch 7/20
136/136 [==============================] - 0s 4ms/step - loss: 2.5998 - accuracy: 0.2077 - val_loss: 2.5776 - val_accuracy: 0.2043
Epoch 8/20
136/136 [==============================] - 0s 3ms/step - loss: 2.5679 - accuracy: 0.2351 - val_loss: 2.5562 - val_accuracy: 0.2399
Epoch 9/20
136/136 [==============================] - 0s 3ms/step - loss: 2.5535 - accuracy: 0.2356 - val_loss: 2.5467 - val_accuracy: 0.2399
Epoch 10/20
136/136 [==============================] - 0s 3ms/step - loss: 2.5469 - accuracy: 0.2356 - val_loss: 2.5418 - val_accuracy: 0.2399
Epoch 11/20
136/136 [==============================] - 0s 3ms/step - loss: 2.5445 - accuracy: 0.2353 - val_loss: 2.5397 - val_accuracy: 0.2395
Epoch 12/20
136/136 [==============================] - 0s 3ms/step - loss: 2.5424 - accuracy: 0.2356 - val_loss: 2.5388 - val_accuracy: 0.2399
Epoch 13/20
136/136 [==============================] - 0s 3ms/step - loss: 2.5412 - accuracy: 0.2352 - val_loss: 2.5380 - val_accuracy: 0.2399
Epoch 14/20
136/136 [==============================] - 0s 3ms/step - loss: 2.5399 - accuracy: 0.2358 - val_loss: 2.5376 - val_accuracy: 0.2399
Epoch 15/20
136/136 [==============================] - 0s 3ms/step - loss: 2.5393 - accuracy: 0.2357 - val_loss: 2.5372 - val_accuracy: 0.2399
Epoch 16/20
136/136 [==============================] - 1s 4ms/step - loss: 2.5393 - accuracy: 0.2360 - val_loss: 2.5370 - val_accuracy: 0.2399
Epoch 17/20
136/136 [==============================] - 1s 5ms/step - loss: 2.5389 - accuracy: 0.2355 - val_loss: 2.5367 - val_accuracy: 0.2399
Epoch 18/20
136/136 [==============================] - 1s 6ms/step - loss: 2.5387 - accuracy: 0.2355 - val_loss: 2.5366 - val_accuracy: 0.2399
Epoch 19/20
136/136 [==============================] - 1s 5ms/step - loss: 2.5385 - accuracy: 0.2357 - val_loss: 2.5368 - val_accuracy: 0.2399
Epoch 20/20
136/136 [==============================] - 1s 4ms/step - loss: 2.5384 - accuracy: 0.2360 - val_loss: 2.5364 - val_accuracy: 0.2399
85/85 [==============================] - 0s 2ms/step - loss: 2.5529 - accuracy: 0.2337
Test accuracy: 0.2337278127670288
```

```
dfS = one_hot_encode_top(dfS, 'source',3)
dfS = one_hot_encode_top(dfS, 'producers',4)
dfS = one_hot_encode_top(dfS, 'studio',4)
dfS = one_hot_encode_top(dfS, 'genre',4)


dfS
```

	episodes	rank	popularity	members	favorites	reviewers	source_Manga	source_Original	source_Other	source_Unknown	...
0	26.0	26	39	795733	43460	405664	0	1	0	0	...
1	1.0	164	449	197791	776	120243	0	1	0	0	...
2	26.0	255	146	408548	10432	212537	1	0	0	0	...
3	26.0	2371	1171	79397	537	32837	0	1	0	0	...
4	52.0	3544	3704	11708	14	4894	1	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
13553	1.0	10776	15213	38	0	21	0	1	0	0	...
13554	1.0	11138	15293	29	0	17	0	1	0	0	...
13556	1.0	13398	15346	19	0	7	0	1	0	0	...
13590	40.0	11345	15227	36	0	2	0	1	0	0	...
13625	12.0	13640	15271	45	0	6	0	0	1	0	...

13518 rows × 23 columns

```
# The accuracy was not good for both of them
# So I set 'Other genre' for Null values in genre to seperete them

   # Predicting Score
X = dfS.to_numpy()
y = yS.to_numpy()
```

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)


model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='linear')  # Linear activation for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model on the test data
loss, mean_absolute_error = model.evaluate(X_test, y_test)
print(f'Mean Absolute Error: {mean_absolute_error}')
```

```
Epoch 1/10
136/136 [==============================] - 2s 6ms/step - loss: 19503012.0000 - mean_absolute_error: 895.9431 - val_loss: 47150.8359
Epoch 2/10
136/136 [==============================] - 1s 4ms/step - loss: 43643.3164 - mean_absolute_error: 132.1529 - val_loss: 30092.8184 - v
Epoch 3/10
136/136 [==============================] - 1s 4ms/step - loss: 28433.0078 - mean_absolute_error: 114.2366 - val_loss: 34882.6094 - v
Epoch 4/10
136/136 [==============================] - 1s 5ms/step - loss: 23105.0762 - mean_absolute_error: 102.6687 - val_loss: 17947.5137 - v
Epoch 5/10
136/136 [==============================] - 1s 6ms/step - loss: 18029.4023 - mean_absolute_error: 91.3569 - val_loss: 14505.7969 - va
Epoch 6/10
136/136 [==============================] - 1s 5ms/step - loss: 14294.6172 - mean_absolute_error: 80.6648 - val_loss: 14284.1113 - va
Epoch 7/10
136/136 [==============================] - 1s 5ms/step - loss: 13975.3916 - mean_absolute_error: 76.0422 - val_loss: 11265.8379 - va
Epoch 8/10
136/136 [==============================] - 1s 5ms/step - loss: 11013.5508 - mean_absolute_error: 70.2625 - val_loss: 10094.6729 - va
Epoch 9/10
136/136 [==============================] - 1s 4ms/step - loss: 10508.1807 - mean_absolute_error: 68.0491 - val_loss: 16520.1152 - va
Epoch 10/10
136/136 [==============================] - 1s 5ms/step - loss: 9417.1133 - mean_absolute_error: 65.3244 - val_loss: 9143.0898 - val_
85/85 [==============================] - 0s 2ms/step - loss: 9109.6514 - mean_absolute_error: 63.7593
Mean Absolute Error: 63.759307861328125
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
modelScore = LinearRegression()

# Fit the model
modelScore.fit(X_train, y_train)

# Make predictions on the test data
y_pred = modelScore.predict(X_test)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)

# Print the MAE
print(f'Mean Absolute Error: {mae}')
```

```
Mean Absolute Error: 0.3744589384681957
```

```
data = nullScores

categorical_columns = data.select_dtypes(include=['object']).columns

numerical_columns = data.select_dtypes(include=['number']).columns

data[categorical_columns] = data[categorical_columns].fillna('Other')

# Fill null values in numerical columns with 0
data[numerical_columns] = data[numerical_columns].fillna(0)
data
```

	source	genre	producers	studio	episodes	rank	popularity	members	fa
11710	Original	Music	Other	Other	1.0	13171	15442	221	
11739	Original	Adventure	Other	Other	1.0	13162	15325	17	
11898	Unknown	Action	Other	Other	26.0	12922	15366	9	
11962	Music	Music	Other	Other	1.0	11405	15409	10	
12090	Light novel	Action	Other	Gonzo	1.0	10933	15381	2168	
...	...	...	...	...	...	...	...	...	
13626	Unknown	Action	Other	Other	54.0	13301	15461	7	
13627	Unknown	Comedy	Other	Other	12.0	13570	15472	7	
13628	Unknown	Fantasy	Other	Other	60.0	13568	15470	7	
13629	Unknown	Fantasy	Other	Other	12.0	13635	15473	7	
13630	Original	Adventure	Other	Other	4.0	11624	15424	10	

```
data = one_hot_encode_top(data, 'source',3)
data = one_hot_encode_top(data, 'producers',4)
data = one_hot_encode_top(data, 'studio',4)
data = one_hot_encode_top(data, 'genre',4)
data
```

	episodes	rank	popularity	members	favorites	reviewers	source_Original	s
11710	1.0	13171	15442	221	0	0	1	
11739	1.0	13162	15325	17	0	0	1	
11898	26.0	12922	15366	9	0	0	0	
11962	1.0	11405	15409	10	0	0	0	
12090	1.0	10933	15381	2168	4	0	0	
...	...	...	...	...	...	...	...	
13626	54.0	13301	15461	7	0	0	0	
13627	12.0	13570	15472	7	0	0	0	
13628	60.0	13568	15470	7	0	0	0	
13629	12.0	13635	15473	7	0	0	0	
13630	4.0	11624	15424	10	0	0	1	

113 rows × 20 columns

```
data = data.to_numpy()
# yPredictions = modelScore.predict(data)
# yPredictions
```

```
# Clustring
from sklearn.cluster import KMeans
# Create a K-Means model with the desired number of clusters
kmeans = KMeans(n_clusters=3, random_state=0)

# Fit the model to your data
kmeans.fit(X)

# Get the cluster labels for each data point
labels = kmeans.labels_

# Get the cluster centers
cluster_centers = kmeans.cluster_centers_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```

```
# Create a dictionary to store data points for each cluster
cluster_data = {}
```

```
for label in set(labels):
    cluster_data[label] = X[labels == label]
cluster_data
```

```
{0: array([[2.6000e+01, 2.3710e+03, 1.1710e+03, ..., 0.0000e+00, 0.0000e+00,
         0.0000e+00],
        [5.2000e+01, 3.5440e+03, 3.7040e+03, ..., 0.0000e+00, 0.0000e+00,
         0.0000e+00],
        [5.2000e+01, 1.1860e+03, 3.1240e+03, ..., 0.0000e+00, 0.0000e+00,
         1.0000e+00],
        ...,
        [1.0000e+00, 1.3398e+04, 1.5346e+04, ..., 0.0000e+00, 1.0000e+00,
         0.0000e+00],
        [4.0000e+01, 1.1345e+04, 1.5227e+04, ..., 0.0000e+00, 0.0000e+00,
         1.0000e+00],
        [1.2000e+01, 1.3640e+04, 1.5271e+04, ..., 0.0000e+00, 0.0000e+00,
         0.0000e+00]]),
 1: array([[ 26.,  26.,  39., ...,   0.,   0.,   0.],
        [220., 705.,  10., ...,   0.,   0.,   0.],
        [  0.,  94.,  36., ...,   0.,   0.,   0.],
        ...,
        [ 10., 193.,  93., ...,   0.,   0.,   0.],
        [ 25.,  42.,  43., ...,   0.,   0.,   0.],
        [ 25.,  50.,  89., ...,   0.,   0.,   0.]]),
 2: array([[1.000e+00, 1.640e+02, 4.490e+02, ..., 0.000e+00, 0.000e+00,
         0.000e+00],
        [2.600e+01, 2.550e+02, 1.460e+02, ..., 0.000e+00, 0.000e+00,
         0.000e+00],
        [2.400e+01, 4.190e+02, 5.360e+02, ..., 1.000e+00, 0.000e+00,
         0.000e+00],
        ...,
        [1.200e+01, 5.200e+01, 5.090e+02, ..., 0.000e+00, 0.000e+00,
         1.000e+00],
        [1.200e+01, 5.972e+03, 5.660e+02, ..., 0.000e+00, 0.000e+00,
         0.000e+00],
        [1.200e+01, 9.630e+02, 5.640e+02, ..., 0.000e+00, 1.000e+00,
         0.000e+00]])}
```