

```
In [1]: 1 # Import necessary packages
2 import zipfile
3 import os
4 import glob
5 import pandas as pd
6 import numpy as np
7 from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
8 from sklearn.preprocessing import MinMaxScaler, StandardScaler
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
11 from sklearn.datasets import make_classification
12 from sklearn.neighbors import KNeighborsClassifier
13 from sklearn.svm import SVC
14 from sklearn.linear_model import LinearRegression
15 import xgboost as xgb
16 import matplotlib.pyplot as plt
17 import math
18 from collections import Counter
19 from tqdm.notebook import tqdm
20 import seaborn as sns
21 from langdetect import DetectorFactory, detect
22 import pickle
23 import warnings
24 from sklearn.decomposition import PCA
```

## Preprocessing Data



```

In [2]:
1 # Initialize
2 DetectorFactory.seed = 0
3 western_lang = ["de", "en", "es", "it", "el", "nl", "da"]
4
5 # Create a unique folder to extract files
6 extracted_folder = "data/namesbystate_extracted"
7 if not os.path.exists(extracted_folder):
8     with zipfile.ZipFile("data/namesbystate.zip", 'r') as zip_ref:
9         zip_ref.extractall(extracted_folder)
10
11 # Read Colorado-specific file
12 co_df = pd.read_csv(os.path.join(extracted_folder, "STATE.CO.txt"), header=None, names=["State", "Sex", "Year", "Name", "Co
13
14 # Aggregate for total count and most popular year in Colorado
15 co_aggregated = co_df.groupby(['Name', 'Sex']).agg({
16     'Count': 'sum',
17     'Year': lambda x: x.value_counts().idxmax()
18 }).reset_index()
19 co_aggregated.rename(columns={'Count': 'CO_Total_Count', 'Year': 'CO_Most_Popular_Year'}, inplace=True)
20
21 # Extracting zip file to a folder named "Extracted"
22 extracted_folder = "data/Extracted"
23
24 # Check if the folder already exists
25 if not os.path.exists(extracted_folder):
26     with zipfile.ZipFile("data/names.zip", 'r') as zip_ref:
27         zip_ref.extractall(extracted_folder)
28
29 # Reading and merging all txt files
30 all_files = glob.glob(os.path.join(extracted_folder, "yob*.txt")) # Adjusted path
31 df_list = []
32
33 for filename in all_files:
34     year = filename[-8:-4] # Extracting the year from filename
35     df = pd.read_csv(filename, names=["Name", "Sex", "Count"])
36     df['Year'] = int(year) # Adding year column
37     df_list.append(df)
38
39 # Concatenating all dataframes
40 all_names = pd.concat(df_list, ignore_index=True)
41
42 # Grouping by 'Name' and 'Sex', then aggregating
43 aggregated_names = all_names.groupby(['Name', 'Sex']).agg({
44     'Count': 'sum',
45     'Year': lambda x: x.value_counts().idxmax() # Most frequent year
46 }).reset_index()
47 aggregated_names
48
49 # Renaming columns for clarity
50 aggregated_names.rename(columns={'Count': 'Total_Count', 'Year': 'Most_Popular_Year'}, inplace=True)
51 aggregated_names['Most_Popular_Year'] = aggregated_names['Most_Popular_Year'].fillna(0).astype(int)
52 aggregated_names.sort_values(by='Total_Count', ascending=False)
53
54 # Getting the most recent year in CO data
55 most_recent_year_CO = co_df['Year'].max()
56
57 # Filtering the most recent year data and aggregating counts by Name and Sex
58 co_recent_counts = co_df[co_df['Year'] == most_recent_year_CO].groupby(['Name', 'Sex'])['Count'].sum().reset_index()
59 co_recent_counts.rename(columns={'Count': 'Count_CO_Recent'}, inplace=True)
60 co_recent_counts.sort_values(by='Count_CO_Recent', ascending=False)
61
62 # Getting the most recent year in the original dataset
63 most_recent_year_all = aggregated_names['Most_Popular_Year'].max()
64
65 # Aggregated names should have a 'Year' column populated to filter
66 all_recent_counts = (all_names[all_names['Year'] == most_recent_year_all]
67     .rename(columns={'Count': 'Count_All_Recent'})
68     .drop(columns='Year'))
69 all_recent_counts.sort_values(by='Count_All_Recent', ascending=False)
70
71 # Adding counts from the most recent year in CO
72 aggregated_names = pd.merge(aggregated_names, co_recent_counts, on=['Name', 'Sex'], how='left')
73
74 # Adding counts from the most recent year in the original dataset
75 aggregated_names = pd.merge(aggregated_names, all_recent_counts[['Name', 'Sex', 'Count_All_Recent']], on=['Name', 'Sex'], h
76
77 # Filling NA
78 aggregated_names['Count_CO_Recent'] = aggregated_names['Count_CO_Recent'].fillna(0).astype(int)
79 aggregated_names['Count_All_Recent'] = aggregated_names['Count_All_Recent'].fillna(0).astype(int)
80
81 # Reading the names rated by wife
82 rated_boys = pd.read_csv("data/baby_name_boy.txt", names=["Name", "Rating", "Sex"])
83 rated_girls = pd.read_csv("data/baby_name_girl.txt", names=["Name", "Rating", "Sex"])
84 rated_names = pd.concat([rated_boys, rated_girls], ignore_index=True)

```

```

85 rated_names = pd.merge(rated_names, aggregated_names[['Name', 'Sex', 'Total_Count', 'Most_Popular_Year']],
86                        on=['Name', 'Sex'], how='left')
87 rated_names['Total_Count'] = rated_names['Total_Count'].fillna(0).astype(int)
88 rated_names['Most_Popular_Year'] = rated_names['Most_Popular_Year'].fillna(0).astype(int)
89
90 # Merge Colorado-specific data into your existing dataframes
91 aggregated_names = pd.merge(aggregated_names, co_aggregated, on=['Name', 'Sex'], how='left')
92 rated_names = pd.merge(rated_names, co_aggregated, on=['Name', 'Sex'], how='left')
93
94 rated_names = pd.merge(rated_names, co_recent_counts, on=['Name', 'Sex'], how='left')
95 rated_names = pd.merge(rated_names, all_recent_counts, on=['Name', 'Sex'], how='left')
96
97 # Fill NA values for the new columns
98 for df in [aggregated_names, rated_names]:
99     df.loc[df['CO_Most_Popular_Year'].isna(), 'CO_Most_Popular_Year'] = df['Most_Popular_Year']
100     df['Count_All_Recent'] = df['Count_All_Recent'].fillna(0).astype(int)
101     df['Count_CO_Recent'] = df['Count_CO_Recent'].fillna(0).astype(int)
102     df['CO_Total_Count'] = df['CO_Total_Count'].fillna(0).astype(int)
103     df.loc[df['CO_Most_Popular_Year'].isna(), 'CO_Most_Popular_Year'] = df['Most_Popular_Year']
104     df['CO_Most_Popular_Year'] = df['CO_Most_Popular_Year'].astype(int)
105
106 aggregated_names = aggregated_names[['Name', 'Sex', 'Total_Count', 'Count_All_Recent', 'Most_Popular_Year', 'CO_Total_Count']]
107 rated_names = rated_names[['Name', 'Sex', 'Total_Count', 'Count_All_Recent', 'Most_Popular_Year', 'CO_Total_Count', 'Count_

```

In [3]:

```

1  tqdm.pandas()
2
3  # Function to save DataFrame to a pickle file
4  def save_to_pickle(df, filename):
5      with open(filename, 'wb') as f:
6          pickle.dump(df, f)
7
8  # Function to load DataFrame from a pickle file
9  def load_from_pickle(filename):
10     with open(filename, 'rb') as f:
11         return pickle.load(f)
12
13  def syllable_count(word):
14     vowels = "aeiouy"
15     word = word.lower()
16     count = 0
17     if word[0] in vowels:
18         count += 1
19     for index in range(1, len(word)):
20         if word[index] in vowels and word[index - 1] not in vowels:
21             count += 1
22     if word.endswith("e"):
23         count -= 1
24     if count == 0:
25         count += 1
26     return count
27
28  def entropy(string):
29     p, lns = Counter(string), float(len(string))
30     return -sum(count/lns * math.log(count/lns, 2) for count in p.values())
31
32  def feature_engineering(df):
33     # Existing features
34     df['Length'] = df['Name'].progress_apply(len)
35     df['Vowels'] = df['Name'].progress_apply(lambda x: sum(1 for char in x.lower() if char in 'aeiou'))
36     df['Consonants'] = df['Length'] - df['Vowels']
37     df['Sex'] = df['Sex'].progress_apply(lambda x: 1 if x == 'M' else 0)
38     df['Has_Repeating'] = df['Name'].progress_apply(lambda x: 1 if max(Counter(x).values()) > 1 else 0)
39     df['Ends_with_Vowel'] = df['Name'].progress_apply(lambda x: 1 if x[-1].lower() in 'aeiou' else 0)
40     df['Starts_with_Vowel'] = df['Name'].progress_apply(lambda x: 1 if x[0].lower() in 'aeiou' else 0)
41     df['Syllable_Count'] = df['Name'].progress_apply(syllable_count)
42     df['Name_Entropy'] = df['Name'].progress_apply(entropy)
43     df['Language_Root'] = df['Name'].progress_apply(lambda x: 1 if detect(x) in western_lang else 0)
44     return df

```

```
In [4]: 1 # Define the name of the saved file
2 saved_filename = 'data/all_names.pkl'
3
4 with warnings.catch_warnings():
5     warnings.simplefilter("ignore")
6
7     # Check if the saved file exists
8     if os.path.exists(saved_filename):
9         print("Loading data from saved file.")
10        all_names = load_from_pickle(saved_filename)
11    else:
12        print("File not found. Running the feature engineering process.")
13
14        # Run your existing code to create 'all_names'
15        all_names = feature_engineering(aggregated_names)
16
17        # Save 'all_names' to a file for future use
18        save_to_pickle(all_names, saved_filename)
19        rated_names = feature_engineering(rated_names)
20
21 feature_cols = [col for col in all_names.columns if col != 'Name']
```

Loading data from saved file.

```
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
0%|          | 0/2142 [00:00<?, ?it/s]
```

```
In [5]: 1 X = rated_names[feature_cols].values
2 y = rated_names["Rating"].values
3 y = y - 1
4
5 class_names = np.sort(pd.DataFrame(y)[0].unique())
6
7 # Split the data into training and test sets
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
9
10 # Feature scaling
11 scaler = StandardScaler()
12 X_train = scaler.fit_transform(X_train)
13 X_test = scaler.transform(X_test)
14
15 # Set up cross-validation
16 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

## Models

```
In [6]: 1 def plot_percentage_conf_matrix(conf_matrix, class_labels):
2     row_sums = conf_matrix.sum(axis=1)
3     norm_conf_matrix = conf_matrix / row_sums[:, np.newaxis]
4     percentage_matrix = np.round(norm_conf_matrix * 100, 2)
5
6     sns.set(font_scale=1.2)
7     plt.figure(figsize=(10, 8))
8     sns.heatmap(percentage_matrix, annot=True, fmt='.2f', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
9
10    plt.title("Confusion Matrix (Percentage)")
11    plt.xlabel("Predicted Labels")
12    plt.ylabel("True Labels")
13    plt.show()
```

# XGBoost

```
In [7]: 1 # Create the XGBoost model
2 xgb_model = xgb.XGBClassifier(objective='multi:softmax', num_class=len(np.unique(y)))
3
4 # Set up hyperparameter grid for search
5 xgb_param_grid = {
6     'n_estimators': [50, 100, 150, 250],
7     'learning_rate': [.01, 0.1, 0.2],
8     'max_depth': [3, 4, 5, 6]
9 }
10
11 # Set up GridSearchCV
12 xgb_grid_search = GridSearchCV(estimator=xgb_model,
13                                param_grid=xgb_param_grid,
14                                cv=cv,
15                                scoring='f1_weighted',
16                                verbose=1)
17
18 # Fit the model
19 xgb_grid_search.fit(X_train, y_train)
20
21 # Get the best parameters
22 xgb_best_params = xgb_grid_search.best_params_
23 print(f"Best xgb parameters: {xgb_best_params}")
24
25 # Evaluate the model with best parameters
26 xgb_best_model = xgb_grid_search.best_estimator_
27 xgb_y_pred = xgb_best_model.predict(X_test)
28
29 # Evaluation metrics
30 xgb_accuracy = accuracy_score(y_test, xgb_y_pred)
31 xgb_f1 = f1_score(y_test, xgb_y_pred, average='macro')
32 xgb_conf_matrix = confusion_matrix(y_test, xgb_y_pred)
33
34 print(f"XGBoost Accuracy: {xgb_accuracy}")
35 print(f"XGBoost F1 Score: {xgb_f1}")
36 plot_percentage_conf_matrix(xgb_conf_matrix, class_names)
```

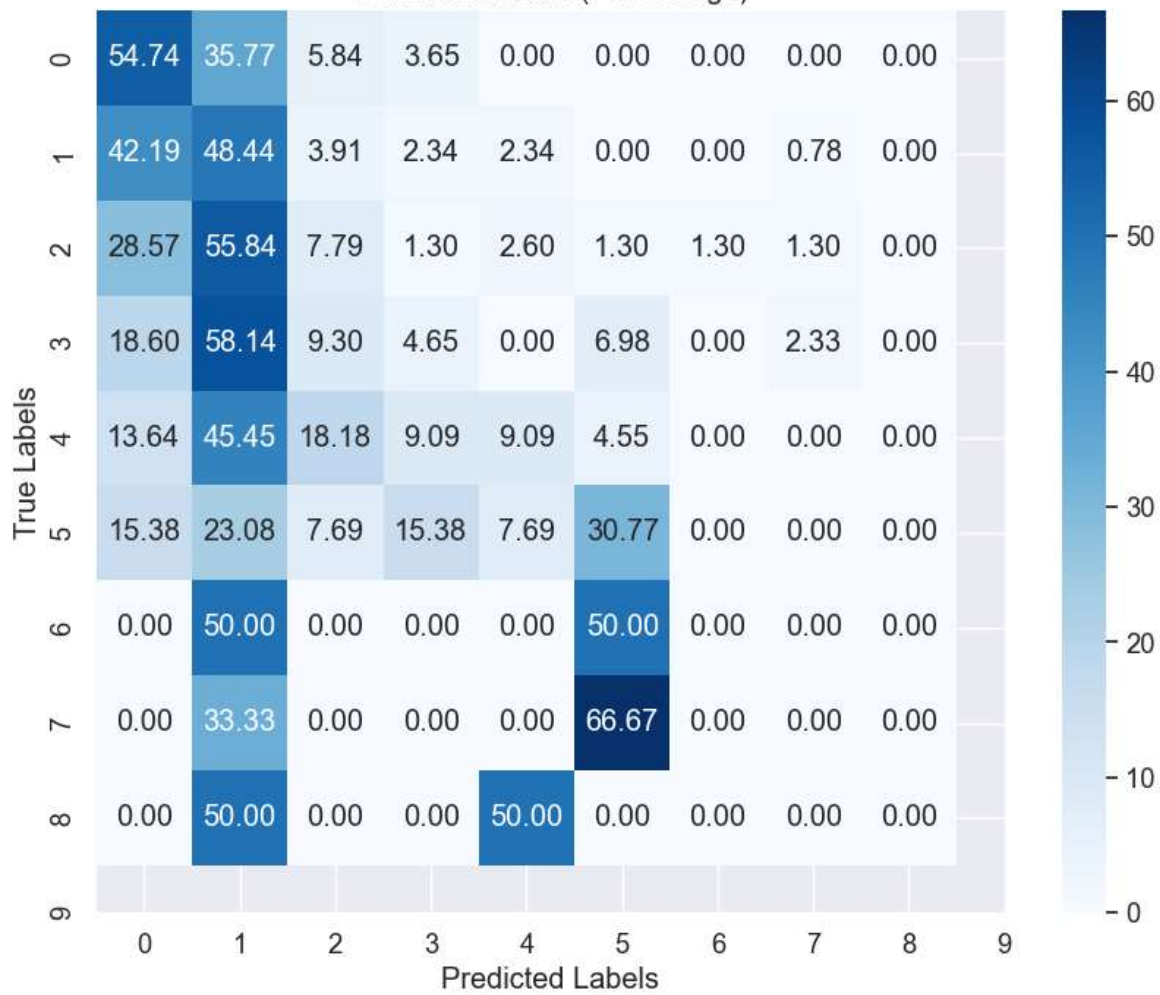
Fitting 5 folds for each of 48 candidates, totalling 240 fits

C:\Users\clopt\AppData\Local\anaconda3\envs\pytorch\lib\site-packages\sklearn\model\_selection\\_split.py:725: UserWarning: The least populated class in y has only 2 members, which is less than n\_splits=5.  
warnings.warn(

Best xgb parameters: {'learning\_rate': 0.1, 'max\_depth': 4, 'n\_estimators': 50}  
XGBoost Accuracy: 0.351981351981352  
XGBoost F1 Score: 0.16678119078872525

Confusion matrix for the 10-class CIFAR-10 dataset. The matrix shows counts for True Labels (rows) vs Predicted Labels (columns). A color bar on the right indicates the count scale from 0 to 60.

	0	1	2	3	4	5	6	7	8
0	54.74	35.77	5.84	3.65	0.00	0.00	0.00	0.00	0.00
1	42.19	48.44	3.91	2.34	2.34	0.00	0.00	0.78	0.00
2	28.57	55.84	7.79	1.30	2.60	1.30	1.30	1.30	0.00
3	18.60	58.14	9.30	4.65	0.00	6.98	0.00	2.33	0.00
4	13.64	45.45	18.18	9.09	9.09	4.55	0.00	0.00	0.00
5	15.38	23.08	7.69	15.38	7.69	30.77	0.00	0.00	0.00
6	0.00	50.00	0.00	0.00	0.00	50.00	0.00	0.00	0.00
7	0.00	33.33	0.00	0.00	0.00	66.67	0.00	0.00	0.00
8	0.00	50.00	0.00	0.00	50.00	0.00	0.00	0.00	0.00



# Random Forest

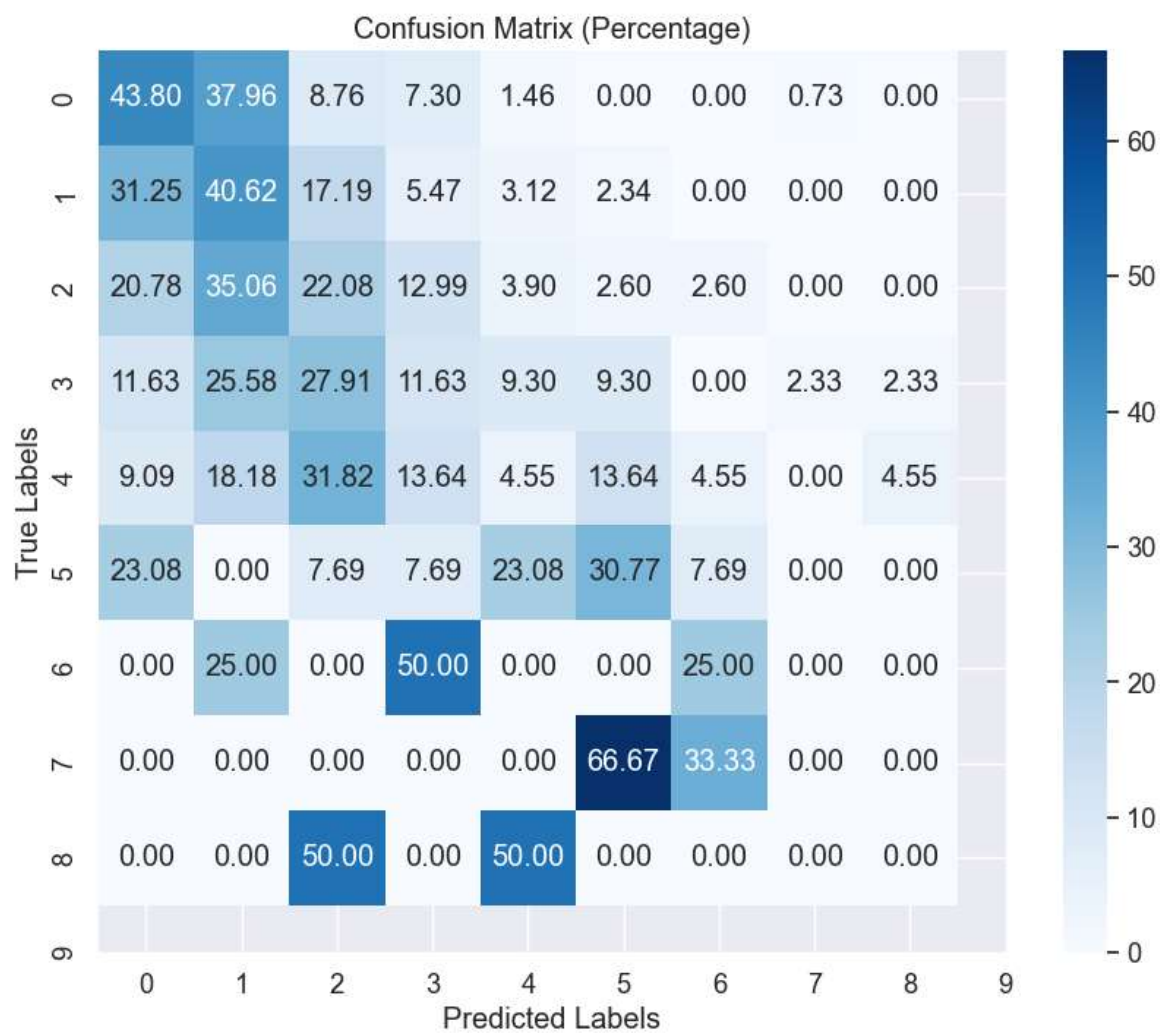
```
In [8]: 1 # Create the random forest model
2 rf_model = RandomForestClassifier(class_weight='balanced')
3
4 # Set up hyperparameter grid for search
5 rf_param_grid = {
6     'n_estimators': [50, 100, 150],
7     'max_depth': [None, 10, 20, 30],
8     'min_samples_split': [2, 5, 10]
9 }
10
11 # Set up GridSearchCV
12 rf_grid_search = GridSearchCV(estimator=rf_model,
13                               param_grid=rf_param_grid,
14                               cv=cv,
15                               scoring='f1_weighted',
16                               verbose=1)
17
18 # Fit the model
19 rf_grid_search.fit(X_train, y_train)
20
21 # Get the best parameters
22 rf_best_params = rf_grid_search.best_params_
23 print(f"Best rf parameters: {rf_grid_search.best_params_}")
24
25 # Evaluate the model with best parameters
26 rf_best_model = rf_grid_search.best_estimator_
27 rf_y_pred = rf_best_model.predict(X_test)
28
29 # Evaluation metrics
30 rf_accuracy = accuracy_score(y_test, rf_y_pred)
31 rf_f1 = f1_score(y_test, rf_y_pred, average='weighted')
32 rf_conf_matrix = confusion_matrix(y_test, rf_y_pred)
33
34 print(f"Random Forest Accuracy: {rf_accuracy}")
35 print(f"Random Forest F1 Score: {rf_f1}")
36 plot_percentage_conf_matrix(rf_conf_matrix, class_names)
```

Fitting 5 folds for each of 36 candidates, totalling 180 fits

C:\Users\clopt\AppData\Local\anaconda3\envs\pytorch\lib\site-packages\sklearn\model\_selection\\_split.py:725: UserWarning: The least populated class in y has only 2 members, which is less than n\_splits=5.  
warnings.warn(

Best rf parameters: {'max\_depth': 20, 'min\_samples\_split': 5, 'n\_estimators': 150}  
Random Forest Accuracy: 0.32634032634032634  
Random Forest F1 Score: 0.32412757322689645





## Support Vector Machine

```
In [9]: 1 # Apply PCA
2 n_components = 10
3 svm_pca = PCA(n_components=n_components)
4 svm_X_train_pca = svm_pca.fit_transform(X_train)
5 svm_X_test_pca = svm_pca.transform(X_test)
```

In [10]:

```
1 # Create the SVM model
2 svm_model = SVC(kernel='rbf', class_weight='balanced')
3
4 # Set up hyperparameter grid for search
5 svm_param_grid = {
6     'C': [0.01, 0.1, 1, 10, 100],
7     'gamma': [0.0001, 0.001, 0.01, 0.1, 1],
8 }
9
10 # Set up GridSearchCV
11 svm_grid_search = GridSearchCV(estimator=svm_model,
12                                param_grid=svm_param_grid,
13                                cv=cv,
14                                scoring='f1_weighted',
15                                verbose=1)
16
17 # Fit the model
18 svm_grid_search.fit(svm_X_train_pca, y_train)
19
20 # Get the best parameters
21 svm_best_params = svm_grid_search.best_params_
22 print(f"Best svm parameters: {svm_grid_search.best_params_}")
23
24 # Evaluate the model with best parameters
25 svm_best_model = svm_grid_search.best_estimator_
26 svm_y_pred = svm_best_model.predict(svm_X_test_pca)
27
28 # Evaluation metrics
29 svm_accuracy = accuracy_score(y_test, svm_y_pred)
30 svm_f1 = f1_score(y_test, svm_y_pred, average='weighted')
31 svm_conf_matrix = confusion_matrix(y_test, svm_y_pred)
32
33 print(f"SVM Accuracy: {svm_accuracy}")
34 print(f"SVM F1 Score: {svm_f1}")
35 plot_percentage_conf_matrix(svm_conf_matrix, class_names)
```

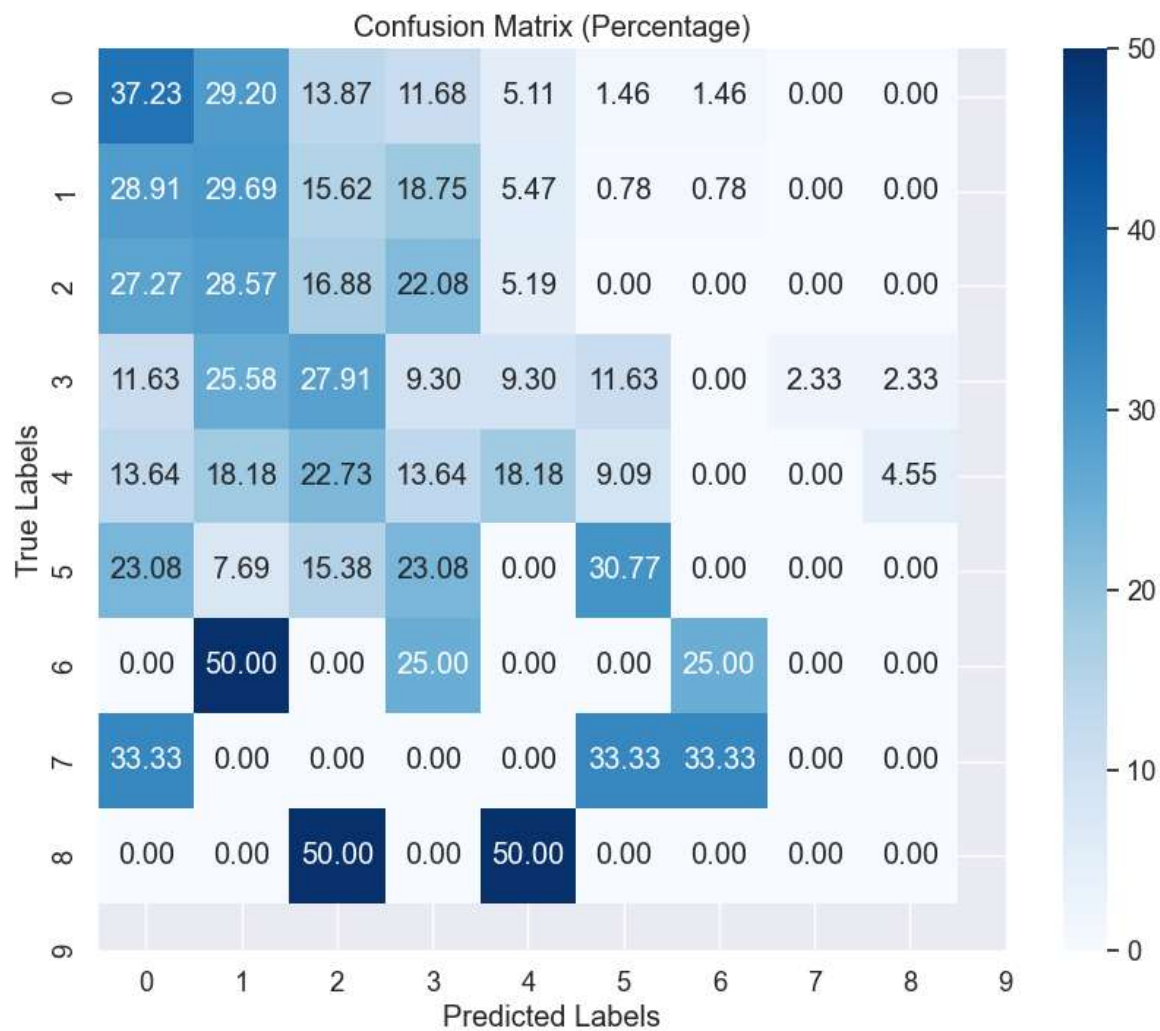
Fitting 5 folds for each of 25 candidates, totalling 125 fits

C:\Users\c\lopt\AppData\Local\anaconda3\envs\pytorch\lib\site-packages\sklearn\model\_selection\\_split.py:725: UserWarning: The least populated class in y has only 2 members, which is less than n\_splits=5.  
warnings.warn(

Best svm parameters: {'C': 100, 'gamma': 0.1}

SVM Accuracy: 0.2680652680652681

SVM F1 Score: 0.27607892352572194



## K-Nearest Neighbors

```
In [11]: 1 # Apply PCA
2 n_components = 10
3 knn_pca = PCA(n_components=n_components)
4 knn_X_train_pca = knn_pca.fit_transform(X_train)
5 knn_X_test_pca = knn_pca.transform(X_test)
```

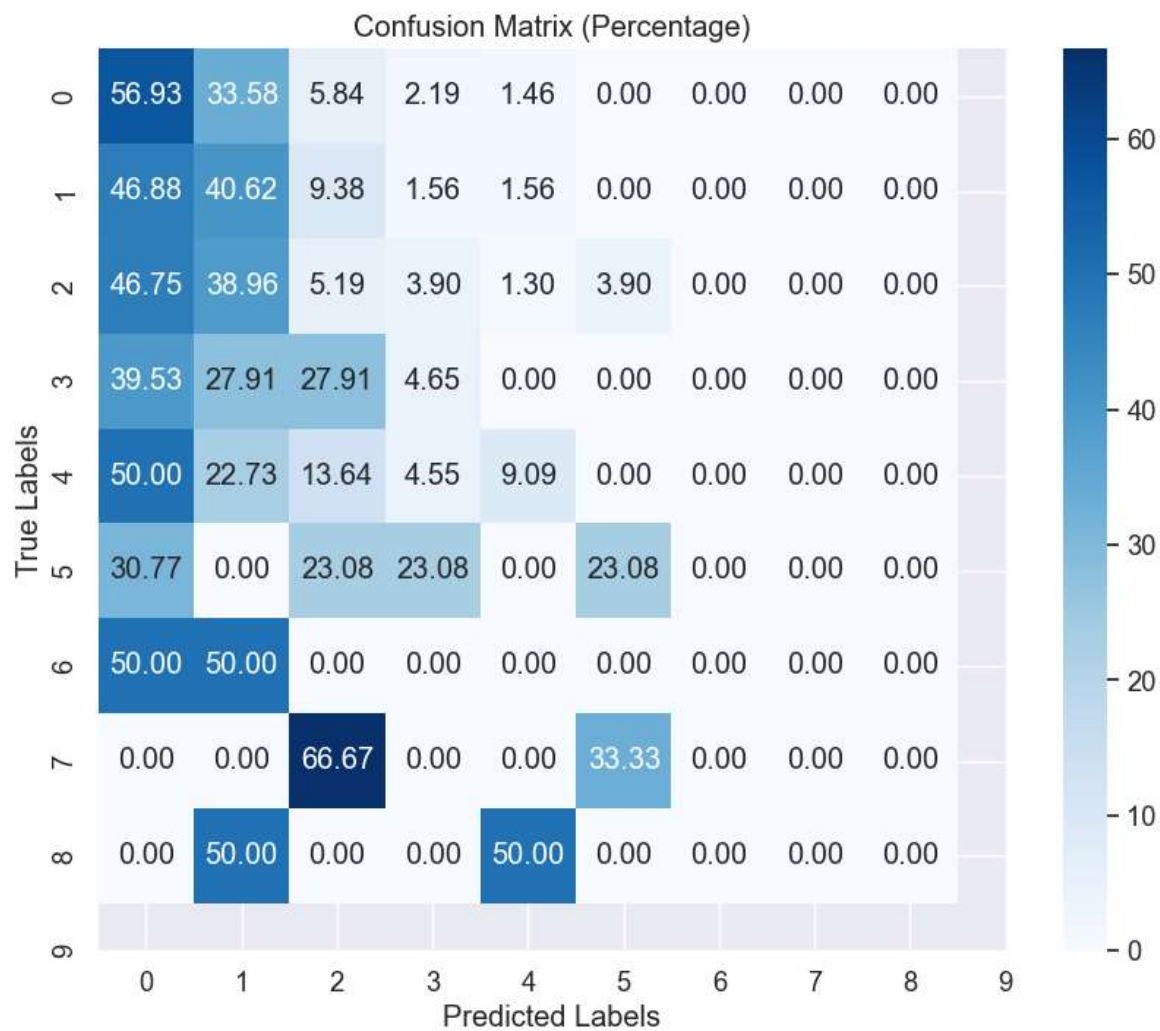
In [12]:

```
1  # Create the KNN model
2  knn_model = KNeighborsClassifier()
3
4  # Set up hyperparameter grid for search
5  knn_param_grid = {
6      'n_neighbors': [3, 5, 7],
7      'weights': ['uniform', 'distance'],
8      'metric': ['euclidean', 'manhattan']
9  }
10
11 # Set up GridSearchCV
12 knn_grid_search = GridSearchCV(estimator=knn_model,
13                                param_grid=knn_param_grid,
14                                cv= cv,
15                                scoring='f1_weighted',
16                                verbose=1)
17
18 # Fit the model
19 knn_grid_search.fit(knn_X_train_pca, y_train)
20
21 # Get the best parameters
22 knn_best_params = knn_grid_search.best_params_
23 print(f"Best svm parameters: {knn_grid_search.best_params_}")
24
25 # Evaluate the model with best parameters
26 knn_best_model = knn_grid_search.best_estimator_
27 knn_y_pred = knn_best_model.predict(knn_X_test_pca)
28
29 # Evaluation metrics
30 knn_y_pred = knn_grid_search.predict(knn_X_test_pca)
31 knn_accuracy = accuracy_score(y_test, knn_y_pred)
32 knn_f1 = f1_score(y_test, knn_y_pred, average='weighted')
33 knn_conf_matrix = confusion_matrix(y_test, knn_y_pred)
34
35 print(f"KNN Accuracy: {knn_accuracy}")
36 print(f"KNN F1 Score: {knn_f1}")
37 plot_percentage_conf_matrix(knn_conf_matrix, class_names)
```

Fitting 5 folds for each of 12 candidates, totalling 60 fits

C:\Users\clopt\AppData\Local\anaconda3\envs\pytorch\lib\site-packages\sklearn\model\_selection\\_split.py:725: UserWarning: The least populated class in y has only 2 members, which is less than n\_splits=5.  
warnings.warn(

Best svm parameters: {'metric': 'euclidean', 'n\_neighbors': 7, 'weights': 'uniform'}  
KNN Accuracy: 0.32867132867132864  
KNN F1 Score: 0.2916584998340771



## Tensorflow Deep Learning

In [13]:

```
1 from tensorflow.keras.callbacks import Callback
2 from IPython.display import display, clear_output
3
4 class SingleLineProgress(Callback):
5     def on_epoch_end(self, epoch, logs=None):
6         metrics_str = " - ".join([f"{key}: {val:.4f}" for key, val in logs.items()])
7         clear_output(wait=True)
8         display(f"Epoch {epoch + 1}: {metrics_str}")
```

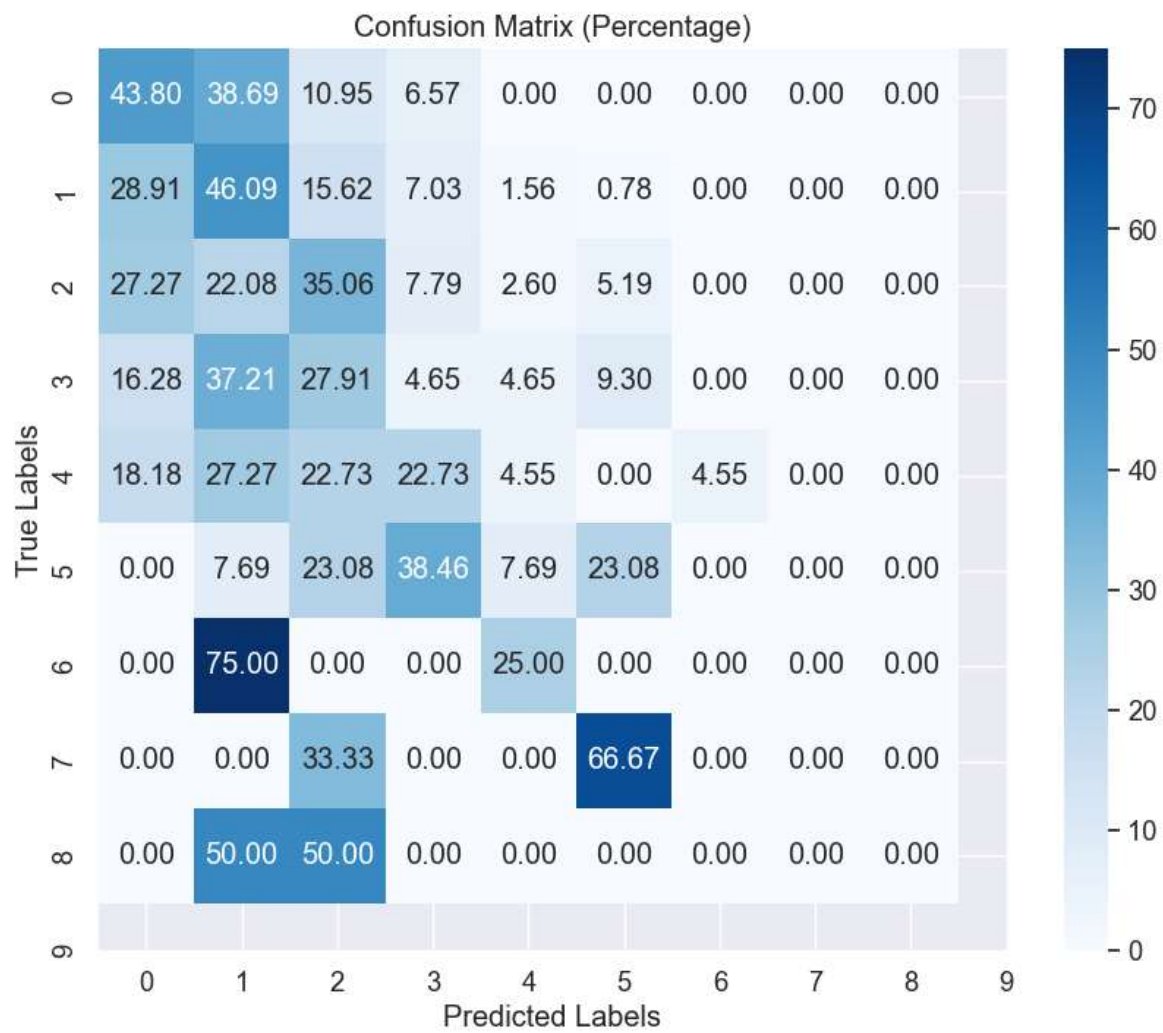
In [14]:

```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Dense
3 from tensorflow.keras.metrics import F1Score
4 from tensorflow.keras.utils import to_categorical
5
6 # Define function for creating the neural network model
7 def create_tf_model(neuron1, neuron2):
8     tf_model = Sequential([
9         Dense(neuron1, activation='relu', input_shape=(X_train.shape[1],)),
10        Dense(neuron2, activation='relu'),
11        Dense(10, activation='softmax') # Assuming 10 classes
12    ])
13    tf_model.compile(optimizer='adam',
14                    loss='categorical_crossentropy',
15                    metrics=[F1Score(average='weighted')])
16    return tf_model
17
18 # One-hot encoding for target labels
19 y_train_one_hot = to_categorical(y_train)
20 y_test_one_hot = to_categorical(y_test)
21
22 # Set up hyperparameter grid for search
23 neuron_combinations = [(16, 16), (32, 16), (32, 32), (64, 32), (64, 64), (64, 128), (128, 128)]
24
25 best_f1 = 0
26 best_neuron_combo = None
27 best_tf_model = None
28
29 for neuron1, neuron2 in neuron_combinations:
30     # Create the neural network model
31     tf_model = create_tf_model(neuron1, neuron2)
32
33     # Fit the model
34     tf_model.fit(X_train,
35                 y_train_one_hot,
36                 epochs=100,
37                 batch_size=16,
38                 verbose=0) # Disable the default progress bar
39
40     # Evaluation metrics
41     tf_y_pred_probs = tf_model.predict(X_test, verbose=0)
42     tf_y_pred = np.argmax(tf_y_pred_probs, axis=1)
43     tf_f1 = f1_score(y_test, tf_y_pred, average='weighted')
44
45     if tf_f1 > best_f1:
46         best_f1 = tf_f1
47         best_neuron_combo = (neuron1, neuron2)
48         best_tf_model = tf_model
49
50 # Use the best model found
51 print(f"Best neuron combination: {best_neuron_combo} with F1 Score: {best_f1}")
52
53 tf_y_pred_probs = best_tf_model.predict(X_test, verbose=0)
54 tf_y_pred = np.argmax(tf_y_pred_probs, axis=1)
55 tf_accuracy = accuracy_score(y_test, tf_y_pred)
56 tf_conf_matrix = confusion_matrix(y_test, tf_y_pred)
57
58 print(f"Tensorflow Accuracy: {tf_accuracy}")
59 print(f"Tensorflow F1 Score: {best_f1}")
60 plot_percentage_conf_matrix(tf_conf_matrix, class_names)
```

Best neuron combination: (32, 16) with F1 Score: 0.34335467539332987

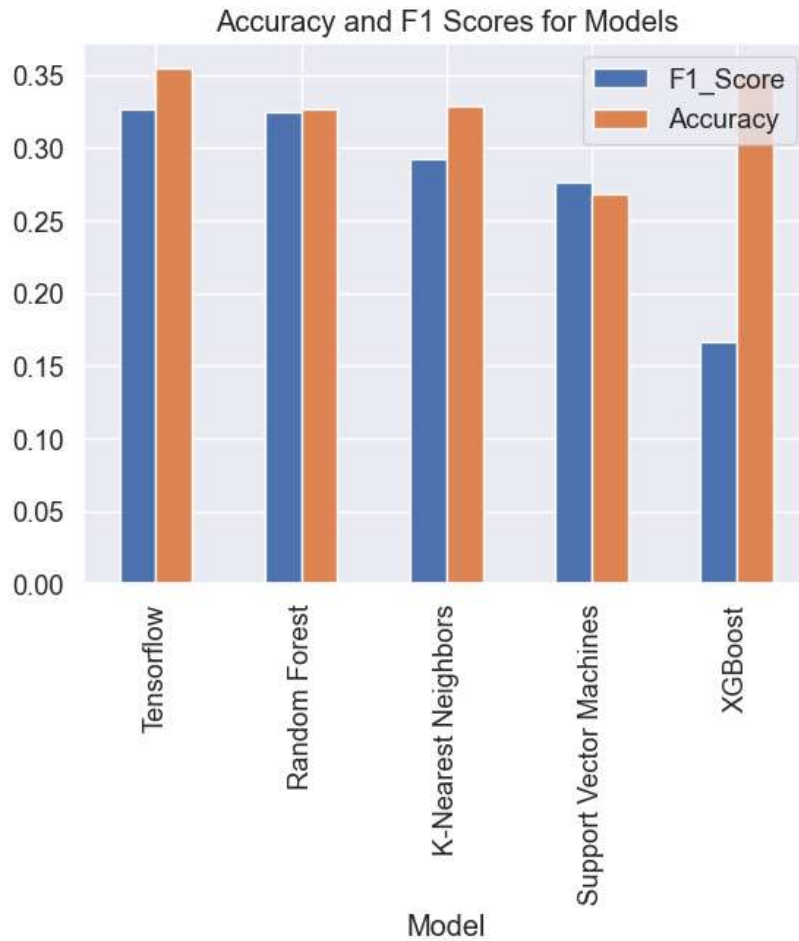
Tensorflow Accuracy: 0.3543123543123543

Tensorflow F1 Score: 0.34335467539332987



## Results

```
In [19]: 1 data = {
2         'Model': ['XGBoost', 'Random Forest', 'Support Vector Machines', 'K-Nearest Neighbors', 'Tensorflow'],
3         'F1_Score': [xgb_f1, rf_f1, svm_f1, knn_f1, tf_f1],
4         'Accuracy': [xgb_accuracy, rf_accuracy, svm_accuracy, knn_accuracy, tf_accuracy]
5     }
6
7 pd.DataFrame(data).sort_values(by='F1_Score', ascending=False).plot(kind='bar', x='Model', title="Accuracy and F1 Scores fo
Out[19]: <Axes: title='{center': 'Accuracy and F1 Scores for Models'}', xlabel='Model'>
```



```
In [16]: 1 # Feature Scaling
2 all_names_features = all_names[feature_cols].values
3 all_names_scaled = scaler.transform(all_names_features)
4
5 # Prediction using trained models
6 rf_predictions_all_names = rf_best_model.predict(all_names_scaled)
7 xgb_predictions_all_names = xgb_best_model.predict(all_names_scaled)
8 svm_predictions_all_names = svm_best_model.predict(svm_pca.transform(all_names_scaled))
9 knn_predictions_all_names = knn_best_model.predict(knn_pca.transform(all_names_scaled))
10 tf_all_names_pred_probs = tf_model.predict(all_names_scaled, verbose=0)
11 tf_predictions_all_names = np.argmax(tf_all_names_pred_probs, axis=1)
12
13 # Add these predictions back to the all_names dataframe
14 all_names['RF_Predicted_Rating'] = rf_predictions_all_names
15 all_names['XGB_Predicted_Rating'] = xgb_predictions_all_names
16 all_names['SVM_Predicted_Rating'] = svm_predictions_all_names
17 all_names['KNN_Predicted_Rating'] = knn_predictions_all_names
18 all_names['Tensorflow_Predicted_Rating'] = tf_predictions_all_names
19 all_names['Average_Predicted_Rating'] = all_names[['RF_Predicted_Rating', 'XGB_Predicted_Rating', 'SVM_Predicted_Rating', '
20
21 # Add 1 back to predictions to scale between 1 and 10
22 columns_to_update = ['RF_Predicted_Rating', 'XGB_Predicted_Rating', 'SVM_Predicted_Rating',
23                     'KNN_Predicted_Rating', 'Tensorflow_Predicted_Rating', 'Average_Predicted_Rating']
24 all_names[columns_to_update] += 1
```

**Find top names**



In [20]:

```
1  # Highlighted rows are names already submitted by my wife
2  def highlight_rows(row):
3      if row['Name'] in rated_names['Name'].values:
4          return ['background-color: lightyellow']*row.shape[0]
5      else:
6          return ['background-color: white']*row.shape[0]
7
8  highlighted_df = (all_names[all_names['Sex']==1]
9                  .sort_values(by='Average_Predicted_Rating', ascending=False)
10                 .reset_index(drop=True)[['Name',
11                                           'RF_Predicted_Rating',
12                                           'XGB_Predicted_Rating',
13                                           'SVM_Predicted_Rating',
14                                           'KNN_Predicted_Rating',
15                                           'Tensorflow_Predicted_Rating',
16                                           'Average_Predicted_Rating']])
17
18     .head(50))
19 highlighted_df.style.apply(highlight_rows, axis=1)
```

Out[20]:

	Name	RF_Predicted_Rating	XGB_Predicted_Rating	SVM_Predicted_Rating	KNN_Predicted_Rating	Tensorflow_Predicted_Rating	Average_Predicted_Rati
0	Nora	10	10	10	2	10	8.4000
1	Maeve	9	9	9	4	9	8.0000
2	Jovie	9	9	9	1	9	7.4000
3	Juliet	8	8	8	4	8	7.2000
4	Eden	8	8	8	3	8	7.0000
5	Ruby	8	8	8	3	8	7.0000
6	Alayna	8	8	8	3	8	7.0000
7	Jade	8	8	8	3	8	7.0000
8	Mckynley	9	3	9	3	9	6.6000
9	Crosslyn	9	3	9	3	9	6.6000
10	Breckynn	9	3	9	3	9	6.6000
11	Macklynn	9	3	9	3	9	6.6000
12	Brezlynn	9	3	9	3	9	6.6000
13	Parklynn	9	3	9	3	9	6.6000
14	Britlynn	9	3	9	3	9	6.6000
15	Braxtynn	9	3	9	3	9	6.6000
16	Langstyn	9	3	9	3	9	6.6000
17	Breklynn	9	3	9	3	9	6.6000
18	Westlynn	9	3	9	3	9	6.6000
19	Rocklynn	9	3	9	3	9	6.6000
20	Synphony	9	6	9	3	5	6.4000
21	Locklynn	9	2	9	3	9	6.4000
22	Lucy	7	7	7	4	7	6.4000
23	Hartlynn	9	2	9	3	9	6.4000
24	Ayla	8	8	8	4	4	6.4000
25	Preslynn	9	2	9	3	9	6.4000
26	Lochlynn	9	2	9	3	9	6.4000
27	Braxlynn	9	1	9	3	9	6.2000
28	Lachlynn	9	1	9	3	9	6.2000
29	Crystell	9	1	9	3	9	6.2000
30	Cathrynn	9	1	9	3	9	6.2000
31	Alina	8	8	8	3	4	6.2000
32	Brynsley	9	1	9	3	9	6.2000
33	Kristynn	9	1	9	3	9	6.2000
34	Ember	9	5	8	1	8	6.2000
35	Ruthlynn	9	1	9	3	9	6.2000
36	Krisslyn	9	1	9	3	9	6.2000
37	Rhettlyn	9	1	9	3	9	6.2000
38	Richlynn	9	1	9	3	9	6.2000
39	Cambrynn	9	1	9	3	9	6.2000
40	Karslynn	9	1	9	3	9	6.2000
41	Decklynn	9	1	9	3	9	6.2000
42	Jahzlynn	9	1	9	3	9	6.2000
43	Brexlynn	9	1	9	3	9	6.2000
44	Becklynn	9	1	9	3	9	6.2000
45	Kirklynn	9	1	9	3	9	6.2000
46	Mckinnly	9	1	9	3	9	6.2000
47	Ivy	6	6	6	6	6	6.0000
48	Lily	7	4	8	3	8	6.0000
49	Charis	7	7	7	2	7	6.0000

