Case Study

Data Analysis on Device Dataset

Muhammad Rasyid Ridho Husein

https://github.com/muhammadrasyidrh/Generated-Device-Dataset

Problem Overview

Objective

Data analysis on device utilization dataset using three different using three different methods

Tool

Python

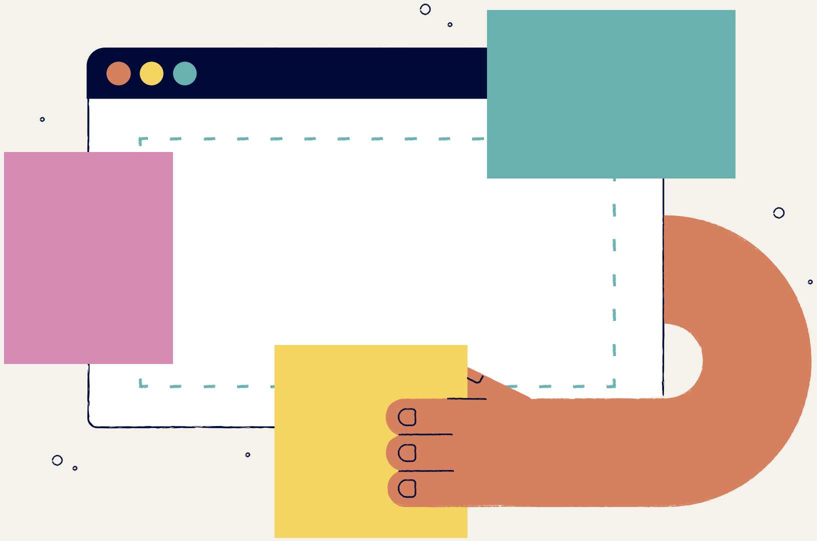
Data

Generated data about device usage data and location type

Dataset Information

Generated dataset consists of 1000 rows and 10 columns

1. User ID : Integer
2. Location : String
3. Login Time : Date Time
4. Name : String
5. Email : String
6. Phone Number : String
7. Birth Year : Integer
8. Cellphone Brand : String
9. Digital Interest : String
10. Location Type : String



Dataset Overview

	User ID	Location	Login Time	Name
0	1	Port Johnathan	2024-01-22 22:42:21.083785	Matthew Ballard
1	2	North Bailey	2024-08-31 15:09:29.617381	Nicole Green PhD
2	3	Ronaldberg	2024-07-22 09:22:54.121453	William Ramos
3	4	West Amanda	2024-08-31 13:54:22.955823	Dr. Michael Meyer
4	5	Sarafort	2024-10-13 19:29:12.397854	Micheal Griffith
5	6	New Jamiebury	2024-03-25 22:14:39.023823	Justin Thomas
6	7	Port Henry	2024-11-20 08:36:34.133092	Stephen Willis
7	8	West Danielfurt	2024-09-23 09:48:05.289213	Candice Turner
8	9	South Leahton	2024-06-26 14:21:46.026850	Richard Smith
9	10	Shelleyport	2024-11-19 17:24:05.512022	Steven Cook

	Email	Phone Number	Birth Year	Cellphone Brand
0	matthew.ballard@outlook.com	389-3270	2002	Oppo
1	nicole.green@yahoo.com	585-3245	2000	Oppo
2	william.ramos@yahoo.com	323-7142	1999	Samsung
3	dr..michael@yahoo.com	784-5647	1986	Vivo
4	micheal.griffith@yahoo.com	565-4150	2003	Huawei
5	justin.thomas@yahoo.com	109-4301	1980	Xiaomi
6	stephen.willis@yahoo.com	162-1852	1990	Xiaomi
7	candice.turner@yahoo.com	294-4551	1996	Xiaomi
8	richard.smith@yahoo.com	993-7612	1993	Oppo
9	steven.cook@outlook.com	560-1618	2000	Oppo

	Digital Interest	Location Type
0	Travel	Urban
1	Shopping	Suburban
2	Music	Rural
3	Tech News	Urban
4	Sport	Suburban
5	Shopping	Rural
6	Gaming	Urban
7	Gaming	Suburban
8	Travel	Rural
9	Sport	Suburban

Dataset Overview

Email Consist of three domains, that are Gmail, Yahoo, and Outlook

Cellphone Brand Consist of six brands, that are Apple, Samsung, Huawei, Oppo, Vivo, and Xiaomi

Digital Interest Consist of eight interests, that are Gaming, Fitness, Social Media, Music, Tech News, Shopping, Travel, and sports

Location Type Consist of three types, that are Urban, Suburban, and Rural

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   User ID           1000 non-null   int64
 1   Location          1000 non-null   object
 2   Login Time        1000 non-null   object
 3   Name              1000 non-null   object
 4   Email             1000 non-null   object
 5   Phone Number      1000 non-null   object
 6   Birth Year        1000 non-null   int64
 7   Cellphone Brand   1000 non-null   object
 8   Digital Interest  1000 non-null   object
 9   Location Type     1000 non-null   object
dtypes: int64(2), object(8)
memory usage: 78.2+ KB
```

Development of New Variables

- Separation of **Name Variable** into **First Name** and **Last Name**
- Separation of **Email Variable** into **Email Username** and **Email Domain**
- Separation of **Phone Number Variable** into **Area Code** and **Local Number**
- Separation of **Login Time Variable** into **Login Hour** and **Login Date**
- Established **Age Variable** using **Birth Year Variable**

```
[ ] df['Age'] = 2024 - df['Birth Year']
```

```
[ ] df['Login Time'] = pd.to_datetime(df['Login Time'])
    df['Login Hour'] = df['Login Time'].dt.hour
```

```
[ ] # Generate Name
    df['First Name'] = df['Name'].apply(lambda x: x.split()[0])
    df['Last Name'] = df['Name'].apply(lambda x: x.split()[1])
```

```
[ ] # Generate Email
    df['Email Username'] = df['Email'].apply(lambda x: x.split('@')[0])
    df['Email Domain'] = df['Email'].apply(lambda x: x.split('@')[1])
```

```
[ ] # Generate Phone Number
    df['Area Code'] = df['Phone Number'].apply(lambda x: x.split('-')[0])
    df['Local Number'] = df['Phone Number'].apply(lambda x: x.split('-')[1])
```

```
[ ] # Generate Login Time
    df['Login Date'] = df['Login Time'].dt.date
```

Confidence Interval

Applied on variables consist of numeric data.

Formula : $CI = mean \pm Z_{\alpha/2} \times \frac{Std.Error}{\sqrt{n}}$

```
[53] # Confidence Interval of Age, Birth Year, and Login Hour
     age_lower, age_upper = confidence_interval_mean(df['Age'], confidence=0.95)
     birthyear_lower, birthyear_upper = confidence_interval_mean(df['Birth Year'], confidence=0.95)
     loginhour_lower, loginhour_upper = confidence_interval_mean(df['Login Hour'], confidence=0.95)

     print(f"95% Confidence Interval for Age: ({age_lower:.2f}, {age_upper:.2f})")
     print(f"95% Confidence Interval for Birth Year: ({birthyear_lower:.2f}, {birthyear_upper:.2f})")
     print(f"95% Confidence Interval for Login Hour: ({loginhour_lower:.2f}, {loginhour_upper:.2f})")
```

```
95% Confidence Interval for Age: (30.95, 31.91)
95% Confidence Interval for Birth Year: (1992.09, 1993.05)
95% Confidence Interval for Login Hour: (11.23, 12.10)
```

Preprocessing Data

Drop unused variables on the models.

```
[ ]  df.drop(['Name','User ID', 'Location', 'Login Time', 'Email', 'Phone Number', 'Birth Year', 'First Name', 'Last Name',
          'Email Username', 'Area Code', 'Local Number', 'Login Date'], axis=1, inplace=True)
```

Remaining variables

```
[ ]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Cellphone Brand   1000 non-null   object
 1   Digital Interest  1000 non-null   object
 2   Location Type     1000 non-null   object
 3   Age               1000 non-null   int64
 4   Login Hour        1000 non-null   int32
 5   Email Domain      1000 non-null   object
dtypes: int32(1), int64(1), object(4)
memory usage: 43.1+ KB
```

Preprocessing Data

Get dummies using **One Hot Encoder** on **Predictor Variables** and **Ordinal Encoder** on **Response Variable**

X = [‘Cellphone Brand’, ‘Digital Interest’, ‘Age’, ‘Login Hour’, ‘Email Domain’]

y = [‘Location Type’]

```
[ ]  X = pd.get_dummies(df, columns=['Cellphone Brand', 'Digital Interest', 'Email Domain']).drop('Location Type', axis=1)
```

```
[ ]  y = df['Location Type']
     ordinal_encoder = OrdinalEncoder(categories=[['Rural', 'Suburban', 'Urban']])
     y = ordinal_encoder.fit_transform(y.values.reshape(-1, 1))
```

Split data into training data (80% of data) and testing data (20% of data)

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ]  X_train.shape, X_test.shape, y_train.shape, y_test.shape

     ((800, 19), (200, 19), (800, 1), (200, 1))
```

Statistical Model

Logistic Regression

Random Forest

Support Vector Machine

Logistic Regression

Training Data

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
    from sklearn.metrics import mean_absolute_error, mean_squared_error, root_mean_squared_error
```

```
[ ] LR = LogisticRegression()
```

```
[ ] LR.fit(X_train, y_train)
```

```
[ ] print('Coef : ', LR.coef_)
    print('Intercept : ', LR.intercept_)
```

```
Coef :  [[ 6.90283082e-03  1.00708679e-04  5.03659858e-02 -6.65259390e-02
  -6.48589094e-02 -6.76759075e-02 -4.10579691e-02  7.47721745e-02
  -5.27431643e-02 -1.98425091e-01 -1.38669242e-01  9.88308782e-02
   4.71910475e-02  5.12556276e-02  1.54187914e-02  6.21605883e-02
   7.65870334e-02 -1.03899071e-01 -8.76685272e-02]
 [-5.45373805e-03 -1.41819573e-02  5.91021188e-02 -3.45886360e-02
   1.35923504e-01  1.06823206e-01  3.53041422e-02 -9.40991538e-02
  -1.60544619e-01  2.44414978e-01  1.44594265e-03  1.20861821e-01
  -1.32532697e-01  3.07818828e-01 -3.60728147e-02 -1.36926259e-01
   4.41485583e-02  3.57553455e-02  1.28561277e-01]
 [-1.44909277e-03  1.40812486e-02 -1.09468105e-01  1.01114575e-01
  -7.10645943e-02 -3.91472985e-02  5.75382694e-03  1.93269792e-02
   2.13287783e-01 -4.59898868e-02  1.37223299e-01 -2.19692699e-01
   8.53416494e-02 -3.59074456e-01  2.06540233e-02  7.47656703e-02
  -1.20735592e-01  6.81437255e-02 -4.08927499e-02]]
Intercept :  [-0.12466047  0.22595597 -0.1012955 ]
```

Logistic Regression

Comparison of Training Data and Testing Data

Accuracy : 0.345

```
Training Data
MAE :   0.79875
MSE :   1.19875
RMSE :   1.0948744220229094
```

```
Testing Data
MAE :   0.855
MSE :   1.255
RMSE :   1.120267825120404
```

```
Training Data
Accuracy :  0.40125

Confusion Matrix :
 [[ 88  99  84]
 [ 68 129  75]
 [ 76  77 104]]

Classification Report :
              precision    recall  f1-score   support

         0.0       0.38      0.32      0.35       271
         1.0       0.42      0.47      0.45       272
         2.0       0.40      0.40      0.40       257

    accuracy                           0.40       800
   macro avg       0.40      0.40      0.40       800
weighted avg       0.40      0.40      0.40       800
```

```
Testing Data
Accuracy :  0.345

Confusion Matrix :
 [[15 22 19]
 [19 30 21]
 [21 29 24]]

Classification Report :
              precision    recall  f1-score   support

         0.0       0.27      0.27      0.27        56
         1.0       0.37      0.43      0.40        70
         2.0       0.38      0.32      0.35        74

    accuracy                           0.34       200
   macro avg       0.34      0.34      0.34       200
weighted avg       0.34      0.34      0.34       200
```

Random Forest

Random Forest using Hyperparameter Tuning to obtain the best parameters.

```
[ ] # Grid Search CV
    param_grid = {
        'n_estimators': [100, 200, 300, 400, 500],
        'max_depth': [3, 5, 7, 10],
        'min_samples_leaf': [5, 10, 25, 50, 100]
    }
```

```
[ ] rf2 = GridSearchCV(estimator=rf1, param_grid=param_grid, cv=5, verbose=1, n_jobs=-1)
```

```
[ ] rf2.fit(X_train, y_train)
```

Best Parameters for Random Forest Model on Training Data

- Max Depth of 10
- Min Samples leaf of 5
- Number of estimators of 400
- Accuracy of 0.351249

Random Forest

Testing Data

```
Testing Data
Accuracy :  0.33

Confusion Matrix :
 [[18 25 13]
 [21 32 17]
 [31 27 16]]

Classification Report :
              precision    recall  f1-score   support

         0.0       0.26      0.32      0.29        56
         1.0       0.38      0.46      0.42        70
         2.0       0.35      0.22      0.27        74

    accuracy                           0.33       200
   macro avg       0.33      0.33      0.32       200
weighted avg       0.33      0.33      0.32       200
```

```
MAE =  0.89
MSE =  1.33
RMSE =  1.1532562594670797
```

Support Vector Machine

Training Data

Support Vector Machine using Hyperparameter Tuning to obtain the best parameters.

```
[ ]  # Grid Search
     param_grid_svc = {
         'C': [100, 10, 1, 0.1, 0.01, 0.001, 0.0001],
         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']
     }
```

```
[ ]  svc2 = GridSearchCV(estimator=svc1, param_grid=param_grid_svc, cv=5, verbose=1, n_jobs=-1)
```

```
[ ]  svc2.fit(X_train, y_train)
```

Best Parameters for Support Vector Machine Model on Training Data

- Regularization Parameter of 1
- Kernel of Polynomial
- Accuracy of 0.365

Support Vector Machine

Testing Data

```
Testing Data
Accuracy :  0.345

Confusion Matrix :
 [[ 6 39 11]
 [11 46 13]
 [10 47 17]]

Classification Report :
              precision    recall  f1-score   support

         0.0       0.22      0.11      0.14        56
         1.0       0.35      0.66      0.46        70
         2.0       0.41      0.23      0.30        74

    accuracy                           0.34       200
   macro avg       0.33      0.33      0.30       200
weighted avg       0.34      0.34      0.31       200
```

```
MAE =  0.76
MSE =  0.97
RMSE =  0.9848857801796105
```

Conclusion

Model	Accuracy	MAE	MSE	RMSE
Logistic Regression	0.345	0.855	1.255	1.12026
Random Forest	0.33	0.89	1.33	1.15325
Support Vector Machine	0.345	0.76	0.97	0.98488

- Overall, the accuracy model is not relatively high, at 34,5%.
- Support Vector Machine model is the optimal model because the model has the lowest error compared to Logistic Regression and Random Forest.
- The Support Vector Machine model can be applied to the generated device data using polynomial kernel function and regularization parameter of 1.

Thanks!

Data Analysis on Device Dataset

Muhammad Rasyid Ridho Husein
muhammadrasyidrh@gmail.com
linkedin.com/in/muhammadrasyidrh
github.com/muhammadrasyidrh