FitBit Analysis

Presented By:

Ankita Indi - 002104957
Nirmiti Patil - 002103927
Saad Ghojaria - 002129718

Abstract

This project describes an analysis of the Fitbit Data. Our objectives are to describe

1. Calories burnt as per daily activity
2. to see active and inactive users
3. Calories burnt as per Our analysis covers three distinct algorithm.

First, we analyze the data and perform cleaning and merging necessary information of the Fitbit data. Next, we perform data transformation, standardization, imputation of data and other feature engineering to make the data more standard. Observe the model and gain insights of the data by getting the accuracy for the algorithm and to find its efficiency. Third, we perform data visualization to demonstrate the observations. We have used multiple libraries to get proper insights and visualization .

INTRODUCTION

Dataset:

https://www.kaggle.com/datasets/nurudeenabdulsalaam/fitbit-fitness-tracker-data?select=dailyActivity_merged.csv

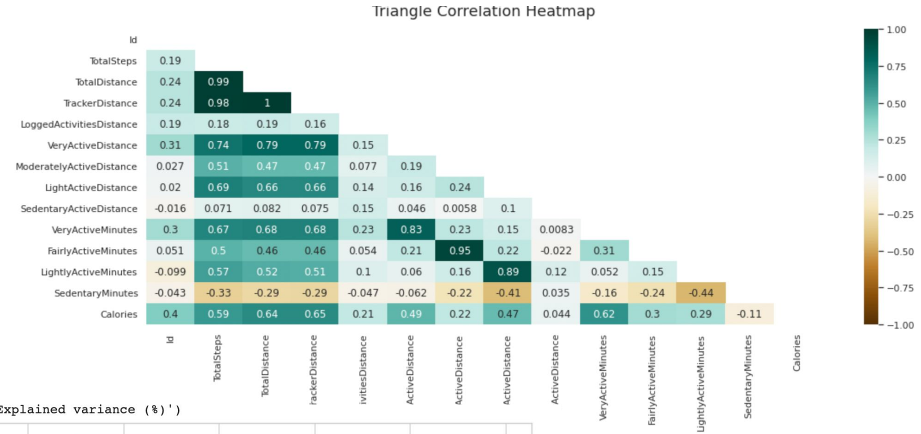
We are using a Fitbit dataset from Kaggle.We have done a comparative study between the models-Linear Regression,Random Forest,XgBoost,K-Means Clustering algorithm.

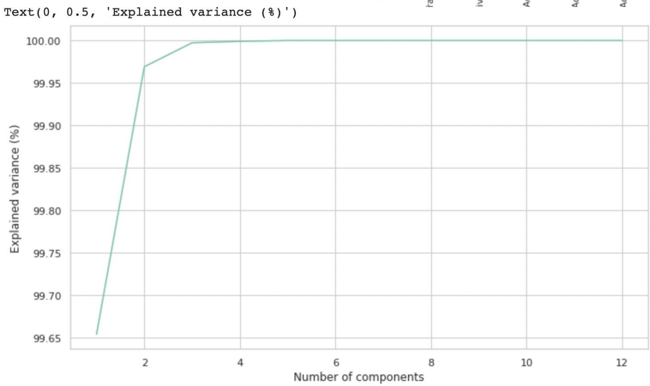
We have also performed Hypertuning using ridge ,lasso ,Elastic-Net Regression for Linear Regression,Elbow method for K-mean,Random Forest ,XGBoost and XGBRegressor

We have split the dataset into training(60%) of the total dataset and the validation and test set are both made up to 20%.We perform some data visualization and infer some insights

Different Data Engineering/Data Profiling methods

1. Data Preprocessing
2. Data Engineering
3. Correlation
4. Standard Scaler
5. Standardization
6. Min Max Scaler
7. Data Imputation - KNN Imputer
8. Extra tree classifier
9. PCA
10. Data Transformation
11. Data Profiling
12. Data Standardization
13. Lasso Regression





METHODS

1. Data Preprocessing :

Below are some highlights of how we processed the data .Once the data was cleaned,We used some Machine learning algorithms like *Random Forest,XgBoost,Linear Regression and K-Means Clustering* and performed hyperparameter tuning

In [625...
```
df_daily_activity.describe()
```

Out[625...

	Id	TotalSteps	TotalDistance	TrackerDistance	LoggedActivitiesDistance	VeryActiveDistance	ModeratelyActiveDistance	LightActiveDistance	SedentaryActiveI
count	9.400000e+02	940.000000	940.000000	940.000000	940.000000	940.000000	940.000000	940.000000	94
mean	4.855407e+09	7637.910638	5.489702	5.475351	0.108171	1.502681	0.567543	3.340819	
std	2.424805e+09	5087.150742	3.924606	3.907276	0.619897	2.658941	0.883580	2.040655	
min	1.503960e+09	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2.320127e+09	3789.750000	2.620000	2.620000	0.000000	0.000000	0.000000	1.945000	
50%	4.445115e+09	7405.500000	5.245000	5.245000	0.000000	0.210000	0.240000	3.365000	
75%	6.962181e+09	10727.000000	7.712500	7.710000	0.000000	2.052500	0.800000	4.782500	
max	8.877689e+09	36019.000000	28.030001	28.030001	4.942142	21.920000	6.480000	10.710000	

In [627...
```
len(df_daily_activity.isnull().sum())
```

Out[627... 15

In [628...
```
df_daily_activity['ActivityDate']=pd.to_datetime(df_daily_activity['ActivityDate'])
```

In [628...

1. Linear Regression Model:

```
# Compute the relevant scores
base_predictions = baseline_y
base_mae = mean_absolute_error(y_valid, base_predictions)
base_mse = mean_squared_error(y_valid, base_predictions)
base_r2 = r2_score(y_valid, base_predictions)
base_errors = abs(base_predictions - y_valid)
base_mape = 100 * np.mean(base_errors / y_valid)
base_accuracy = 100 - base_mape
print('Model Performance')
print('Mean Absolute Error: {:0.4f}.'.format(base_mae))
print('Mean Squared Error: {:0.4f}.'.format(base_mse))
print('R^2 Score = {:0.4f}.'.format(base_r2))
print('Accuracy = {:0.2f}%.'.format(base_accuracy))
```

```
Model Performance
Mean Absolute Error: 649.0120.
Mean Squared Error: 650005.8072.
R^2 Score = -0.0292.
Accuracy = 72.74%.
```

In [65]:
```
regressor = LinearRegression()
mlr = regressor.fit(X_train, y_train)

scoring(mlr, X_valid, y_valid)
```

```
Model Performance
Mean Absolute Error: 571.5923.
Mean Squared Error: 461353.1211.
R^2 Score = 0.2695.
Accuracy = 74.79%.
```

Accuracy for Linear Regression: 74.79%

Linear Regression Model after Hypertuning:

1. Hypertuning using ridge:

```
In [96]: # Train model with default alpha=1
         ridge = Ridge(alpha=10000).fit(X_train, y_train)
         # get cross val scores
         scoring(ridge, X_valid, y_valid)

         Model Performance
         Mean Absolute Error: 573.7368.
         Mean Squared Error: 463473.5759.
         R^2 Score = 0.2662.
         Accuracy = 74.66%.
```

Accuracy for Linear Regression: 74.66%

2. Hypertuning using lasso:

```
In [101]: # Train model with default alpha=1
          elastic_net = ElasticNet(alpha=70, l1_ratio=0.3).fit(X_train, y_train)
          # get cross val scores
          scoring(elastic_net, X_valid, y_valid)

          Model Performance
          Mean Absolute Error: 574.3572.
          Mean Squared Error: 464119.3491.
          R^2 Score = 0.2651.
          Accuracy = 74.63%.
```

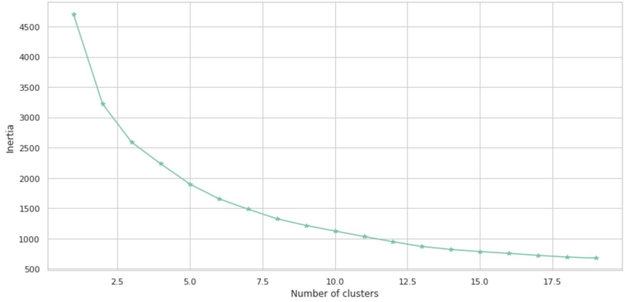
Accuracy for Linear Regression: 74.63%

2. K-Means Clustering Method:

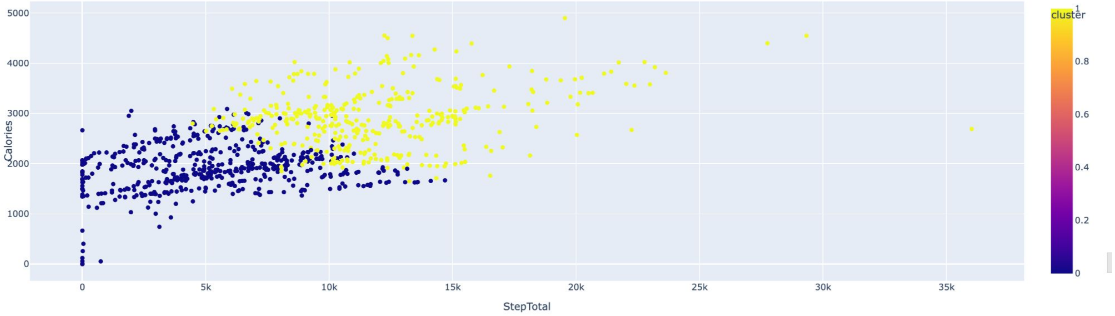
```python
SSE = []
for i in range(1,20):
    kmeans = cluster.KMeans(n_clusters = i, init='k-means++') # iterate from range (1, 20)
    kmeans.fit(data_scaled)
    SSE.append(kmeans.inertia_)

# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker="*")
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

```
Text(0, 0.5, 'Inertia')
```



Visualization



3. XgBoost

```
[685]
## XGBoost Regressor

xgb_regressor = XGBRegressor(random_state=42)
xgb = xgb_regressor.fit(X_train, y_train)

scoring(xgb, X_valid, y_valid)

```

```
[20:19:28] WARNING: /workspace/src/objective/regressi
Model Performance
Mean Absolute Error: 542.0950.
Mean Squared Error: 460604.8582.
R^2 Score = 0.2707.
Accuracy = 76.06%.
```

Hyperparameter Tuning

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.exceptions import FitFailedWarning

xgb_base = XGBRegressor()

xgb_random = RandomizedSearchCV(estimator=xgb_base, param_distributions=xgb_grid,
                                n_iter=200, cv=3, verbose=2,
                               random_state=42, n_jobs=-1)

xgb_random.fit(X_train_temp, y_train_temp)

xgb_random.best_params_

```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
{'tree_method': 'approx',
 'objective': 'reg:squarederror',
 'n_estimators': 1000,
 'min_child_weight': 2,
 'max_depth': 12,
 'gamma': 0,
 'eta ': 0.4}
```

4. Random Forest

Random Forest

```
rf_regressor = rf_sk(random_state=42)
rf = rf_regressor.fit(X_train, y_train)

scoring(rf, X_valid, y_valid)
```

```
Model Performance
Mean Absolute Error: 506.9816.
Mean Squared Error: 407747.2590.
R^2 Score = 0.3544.
Accuracy = 77.49%.
```

```
rf_min_impurity_decrease = [0.0, 0.05, 0.1]
rf_bootstrap = [True, False]

# Create the grid
rf_grid = {'n_estimators': rf_n_estimators,
               'max_depth': rf_max_depth,
               'max_features': rf_max_features,
               'criterion': rf_criterion,
               'min_samples_split': rf_min_samples_split,
               'min_impurity_decrease': rf_min_impurity_decrease,
               'bootstrap': rf_bootstrap}

rf_grid
```

```
{'n_estimators': [200, 400, 600, 800, 1000],
 'max_depth': [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, None],
 'max_features': ['auto', 'sqrt', 'log2'],
 'criterion': ['mse', 'mae'],
 'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],
 'min_impurity_decrease': [0.0, 0.05, 0.1],
 'bootstrap': [True, False]}
```

```
# Tune the Random Forest Model
rf_base = rf_sk()
rf_random = RandomizedSearchCV(estimator = rf_base, param_distributions = rf_grid,
                               n_iter = 200, cv = 3, verbose = 2, random_state = 42,
                               n_jobs = -1)

rf_random.fit(X_train_temp, y_train_temp)

rf_random.best_estimator_
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
RandomForestRegressor(criterion='mae', max_depth=35, min_impurity_decrease=0.1,
                      min_samples_split=5, n_estimators=200)
```

5. Accuracies after Hyper parameter tuning

```
[20:19:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated
```

Out [688...

	RandomForestRegressor(random_state=42)	XGBRegressor(random_state=42)
Mean Absolute Error	471.8517	485.1102
Mean Squared Error	371613.2164	383394.9792
R^2	0.3468	0.3281
Accuracy	80.4681	79.9193