Data Diary Challenge

Machine Learning – A prediction model for selected persona

Selected persona – Michael (Personal Trainer and health-conscious individual)

Identified prediction ideas :

- A model that uses calories burned details of Michael, to predict his future calories burned.
- A model that uses the number of calories burned from the data collected from fit bit to predict the number of calories burned in future and to track progress over time.
- A model that uses data to predict and offer personalized recommendations based on his past workout data

A prediction model to predict calories burned prediction.

Michael's each day comprises with many activities and responsibilities; therefore, he expects an app that can track all his tasks. The planned model can predict the success of his future calories burned that is based on his past data stored in the fit bit.

Prediction model is based on the following attributes or the parameters that can be captured from the Fit bit app.

The given code performs the task of predicting the future calories burned by Michael using different machine learning algorithms. The steps involved in the process are explained below:

Importing the Required Libraries: The necessary libraries required for performing the task are imported, including pandas, numpy, matplotlib, and seaborn.

Loading the Dataset: The dataset containing the required features and the target variable (calories_burned) is loaded using pandas read_csv method.

Exploratory Data Analysis: This step involves analyzing the dataset to obtain a better understanding of its features, shape, size, null values, etc.

First, the first five rows of the dataset are displayed using the head() function. The number of rows and columns are checked using the shape attribute.

Exploratory Data Analysis:

```
[ ] # Displaying the first 5 rows of the dataset
    df.head()
```

	date	weight(lbs)	height(in)	age	activity_level	calories_burned
0	2022-10-01	181	70	27	1.6	2919
1	2022-10-02	168	72	27	2.2	4092
2	2022-10-03	154	74	27	2.2	3498
3	2022-10-04	170	66	27	1.3	2146
4	2022-10-05	164	65	27	2.3	4066

```
# checking the number of rows and columns
df.shape
```

```
(151, 6)
```

The info() method is used to obtain information about the data types of the columns, null values if any, and the memory usage of the dataset.

```
# getting some informations about the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 151 entries, 0 to 150
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   date             151 non-null    object 
 1   weight(lbs)      151 non-null    int64  
 2   height(in)       151 non-null    int64  
 3   age              151 non-null    int64  
 4   activity_level   151 non-null    float64
 5   calories_burned  151 non-null    int64  
dtypes: float64(1), int64(4), object(1)
memory usage: 8.3+ KB
```

The number of duplicate rows is checked using the duplicated() function. Null values are checked using the isnull() method, and the count of null values is displayed.

```
[ ] duplicate_rows_df =df[df.duplicated()]
    print("number of duplicate rows:", duplicate_rows_df.shape)

    number of duplicate rows: (0, 6)
```

```
    # Checking for null values
    df.isnull().sum()

    date               0
    weight(lbs)        0
    height(in)         0
    age                0
    activity_level     0
    calories_burned    0
    dtype: int64
```

The describe() method is used to obtain descriptive statistics about the dataset.

```
[ ] # Describing the dataset
    df.describe()
```

	weight(lbs)	height(in)	age	activity_level	calories_burned
count	151.000000	151.000000	151.0	151.000000	151.000000
mean	167.483444	71.125828	27.0	1.927152	3406.105960
std	14.208380	3.258434	0.0	0.386943	864.382023
min	145.000000	64.000000	27.0	1.300000	1844.000000
25%	157.000000	68.000000	27.0	1.600000	2684.000000
50%	167.000000	72.000000	27.0	2.000000	3516.000000
75%	179.000000	74.000000	27.0	2.200000	4121.000000
max	195.000000	76.000000	27.0	2.500000	5421.000000

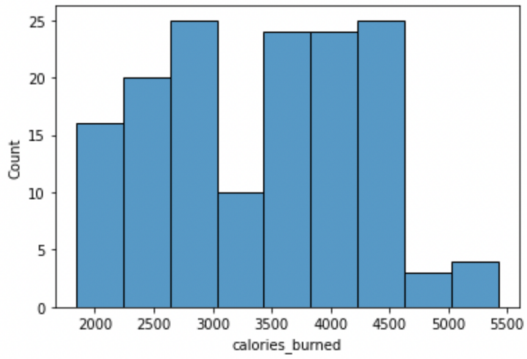
Histograms of the features are plotted using the hist() function, and a correlation matrix is created and plotted using the heatmap() function of the seaborn library. Lastly, a scatter plot of calories_burned vs activity_level is plotted.

```
[ ] print(df.columns)
```

```
Index(['date', 'weight(lbs)', 'height(in)', 'age', 'activity_level',
       'calories_burned'],
      dtype='object')
```

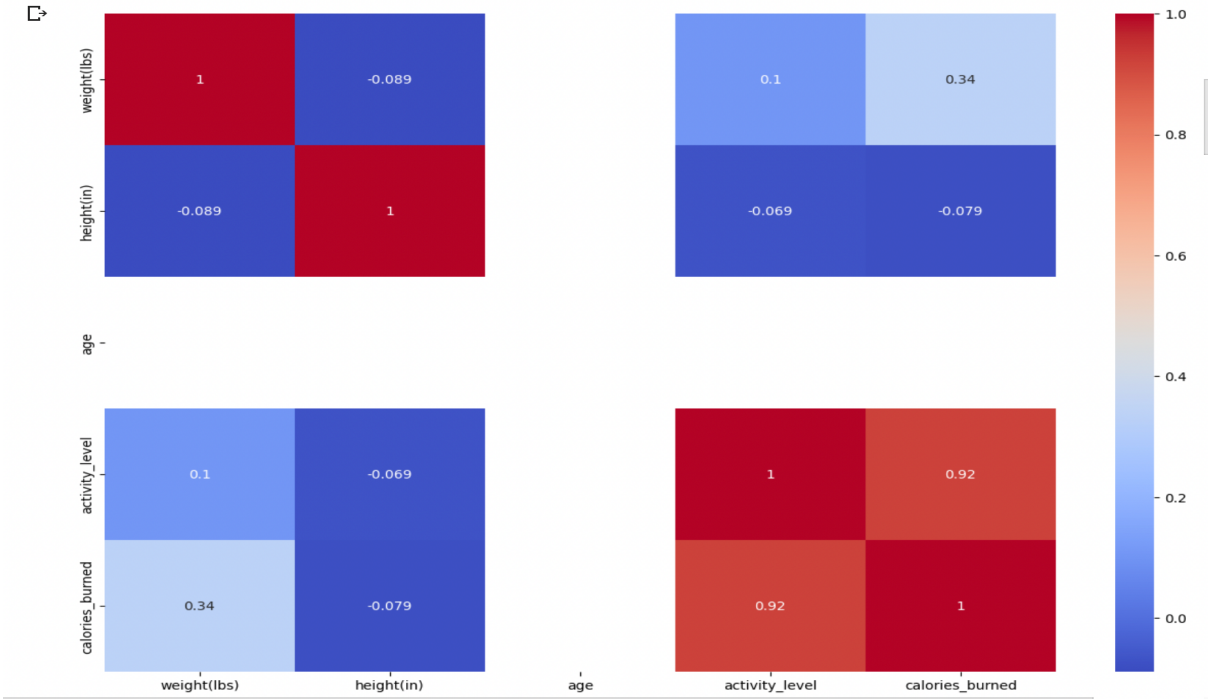
```
sns.histplot(data=df, x="calories_burned")
```

```
<Axes: xlabel='calories_burned', ylabel='Count'>
```

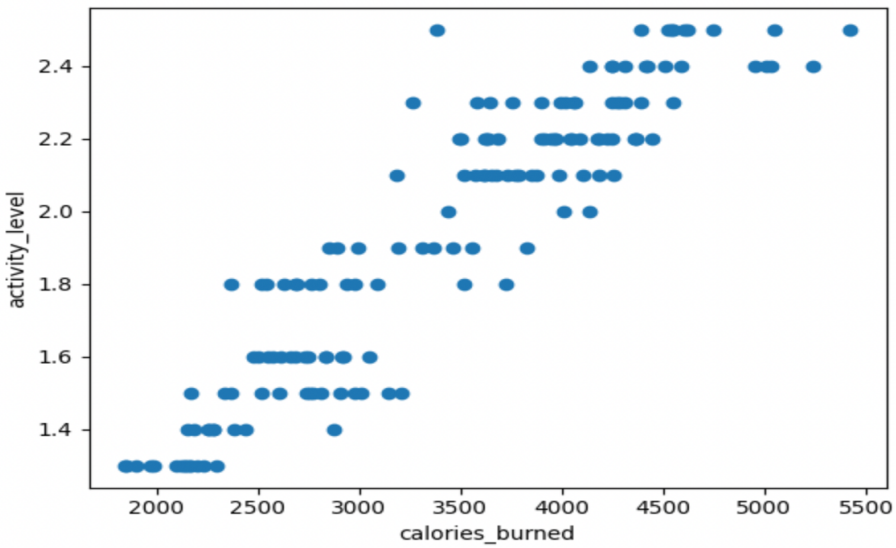


The correlation matrix is plotted using the Seaborn heatmap() function to visualize the correlations between each pair of columns.

```
# Plotting the correlation matrix
plt.figure(figsize=(15,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```

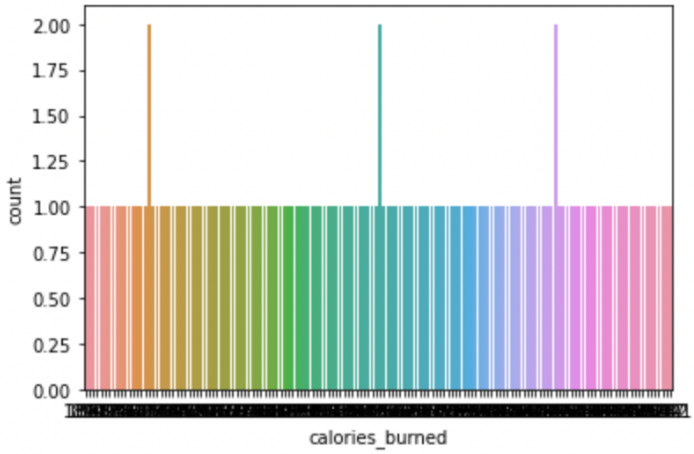


```
# Plotting scatter plot of calories_burned vs activity_level
plt.scatter(x=df.calories_burned, y=df.activity_level)
plt.xlabel("calories_burned")
plt.ylabel("activity_level")
plt.show()
```




```
sns.countplot(x="calories_burned", data=df)
```

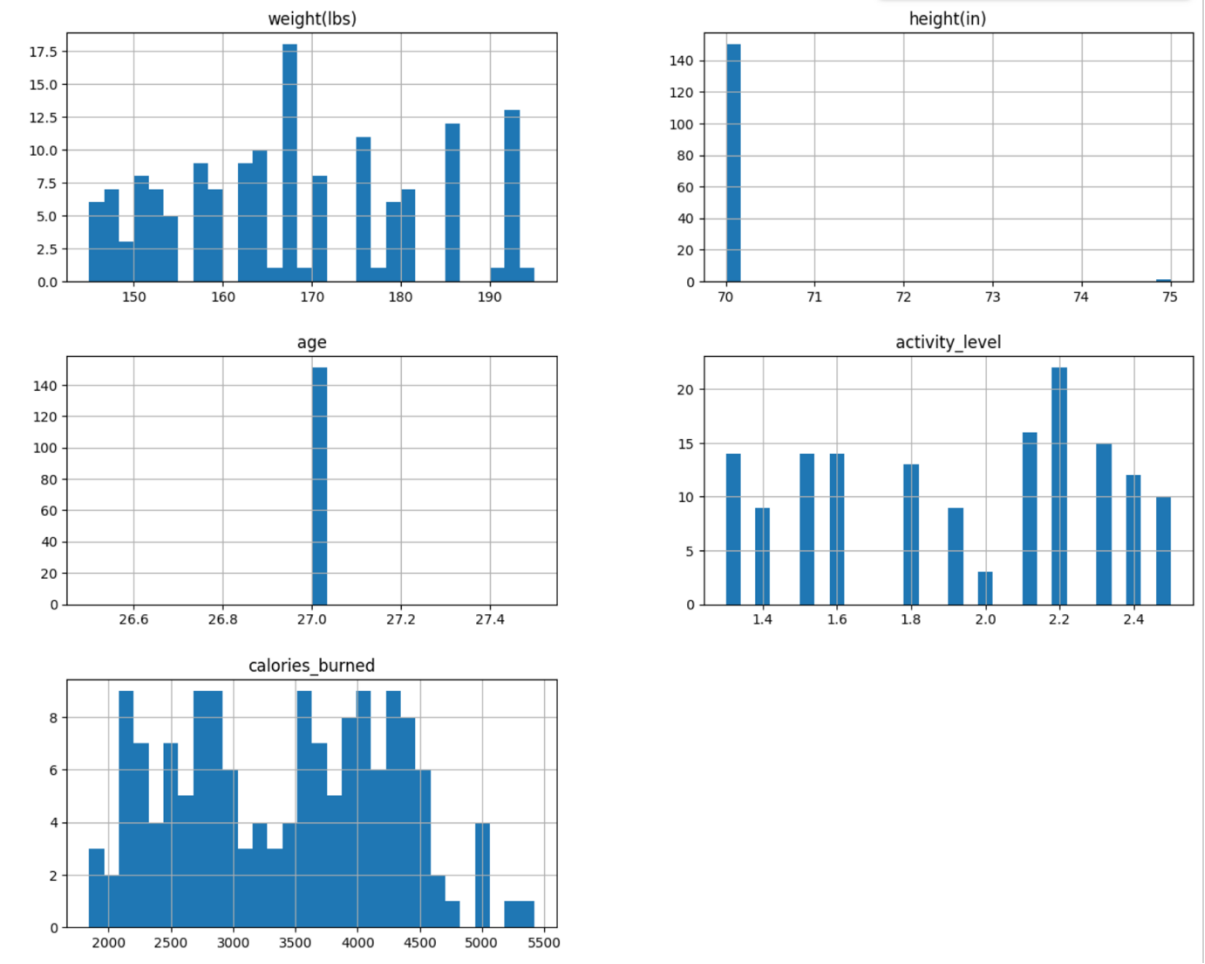
```
<Axes: xlabel='calories_burned', ylabel='count'>
```



Data Visualization:

This step involves visualizing the dataset using different plots and graphs to obtain insights and patterns present in the dataset.

```
# Plotting histogram of all features
df.hist(figsize=(15,12), bins=30)
plt.show()
```



Splitting the Dataset: The dataset is split into a training set and a test set using the train_test_split() function of the sklearn library.

Building Machine Learning Models: Three machine learning models are built, namely Linear Regression, Random Forest Regression, and XGBoost Regression.

The Linear Regression model is built using the LinearRegression() class of the sklearn library.

The Random Forest Regression model is built using the RandomForestRegressor() class, and the XGBoost Regression model is built using the XGBRegressor() class of the sklearn library.

The fit() method is used to train the models, and the predict() method is used to predict the target variable (calories_burned) for the test set.

The mean_squared_error() and r2_score() functions of the sklearn library are used to calculate the mean squared error and R-squared scores of the models.

```
[ ] # Linear Regression
    lr = LinearRegression()
    lr.fit(X_train, y_train)
    lr_pred = lr.predict(X_test)
    lr_mse = mean_squared_error(y_test, lr_pred)
    lr_r2 = r2_score(y_test, lr_pred)
    print("Linear Regression Mean Squared Error:", lr_mse)
    print("Linear Regression R^2 Score:", lr_r2)
```

```
Linear Regression Mean Squared Error: 72526.34879601725
Linear Regression R^2 Score: 0.8819315228569921
```

```
[ ] # Random Forest Regression
    rfr = RandomForestRegressor(n_estimators = 100, random_state = 0)
    rfr.fit(X_train, y_train)
    rfr_pred = rfr.predict(X_test)
    rfr_mse = mean_squared_error(y_test, rfr_pred)
    rfr_r2 = r2_score(y_test, rfr_pred)
    print("Random Forest Regression Mean Squared Error:", rfr_mse)
    print("Random Forest Regression R^2 Score:", rfr_r2)
```

```
Random Forest Regression Mean Squared Error: 82835.75120260213
Random Forest Regression R^2 Score: 0.865148443843551
```

```
[ ] # XGBoost Regression
    xgb = XGBRegressor(n_estimators = 100, random_state = 0)
    xgb.fit(X_train, y_train)
    xgb_pred = xgb.predict(X_test)
    xgb_mse = mean_squared_error(y_test, xgb_pred)
    xgb_r2 = r2_score(y_test, xgb_pred)
    print("XGBoost Regression Mean Squared Error:", xgb_mse)
    print("XGBoost Regression R^2 Score:", xgb_r2)
```

```
XGBoost Regression Mean Squared Error: 89763.04972896076
XGBoost Regression R^2 Score: 0.8538712238910822
```

```
from sklearn.svm import SVR

# SVR model
svr = SVR(kernel = 'rbf')
svr.fit(X_train, y_train)
svr_pred = svr.predict(X_test)
svr_mse = mean_squared_error(y_test, svr_pred)
svr_r2 = r2_score(y_test, svr_pred)
print("SVR Mean Squared Error:", svr_mse)
print("SVR R^2 Score:", svr_r2)
```

```
SVR Mean Squared Error: 620265.8560359823
SVR R^2 Score: -0.00975502368036274
```

```
from sklearn.ensemble import GradientBoostingRegressor

# Gradient Boosting Regression model
gbr = GradientBoostingRegressor(n_estimators = 100, random_state = 0)
gbr.fit(X_train, y_train)
gbr_pred = gbr.predict(X_test)
gbr_mse = mean_squared_error(y_test, gbr_pred)
gbr_r2 = r2_score(y_test, gbr_pred)
print("Gradient Boosting Regression Mean Squared Error:", gbr_mse)
print("Gradient Boosting Regression R^2 Score:", gbr_r2)
```

```
Gradient Boosting Regression Mean Squared Error: 77240.32960457343
Gradient Boosting Regression R^2 Score: 0.8742574493018352
```

Evaluating the Models: The mean squared error and R-squared scores of the models are displayed, which can be used to evaluate the performance of the models.
Based on the given code, the Linear Regression model has the best accuracy with an R-squared score of 0.88.

Several regression models are trained on the training set and evaluated on the test set using the mean squared error (MSE) and R-squared (R^2) metrics. The models used are linear regression, random forest regression, and XGBoost regression.

The StandardScaler function is used to scale the data before training the Support Vector Regression model.

A Support Vector Regression model is trained on the scaled data and evaluated on the test set using MSE and R^2 metrics.

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
# Linear Regression after Feature Scaling
```

```
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
lr_mse = mean_squared_error(y_test, lr_pred)
lr_r2 = r2_score(y_test, lr_pred)
print("Linear Regression Mean Squared Error:", lr_mse)
print("Linear Regression R^2 Score:", lr_r2)
```

```
Linear Regression Mean Squared Error: 81564.56369626606
Linear Regression R^2 Score: 0.8672178596562572
```

```
# Random Forest Regression after Feature Scaling
```

```
rfr = RandomForestRegressor(n_estimators = 100, random_state = 0)
rfr.fit(X_train, y_train)
rfr_pred = rfr.predict(X_test)
rfr_mse = mean_squared_error(y_test, rfr_pred)
rfr_r2 = r2_score(y_test, rfr_pred)
print("Random Forest Regression Mean Squared Error:", rfr_mse)
print("Random Forest Regression R^2 Score:", rfr_r2)
```

```
Random Forest Regression Mean Squared Error: 87688.2413994526
Random Forest Regression R^2 Score: 0.8572488854429913
```

```
# XGBoost Regression after Feature Scaling
```

```
xgb = XGBRegressor(n_estimators = 100, random_state = 0)
xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)
xgb_mse = mean_squared_error(y_test, xgb_pred)
xgb_r2 = r2_score(y_test, xgb_pred)
print("XGBoost Regression Mean Squared Error:", xgb_mse)
print("XGBoost Regression R^2 Score:", xgb_r2)
```

```
XGBoost Regression Mean Squared Error: 109521.36945302448
XGBoost Regression R^2 Score: 0.8217058831638654
```

```
# SVR after Feature Scaling
```

```
svr = SVR(kernel='rbf')
svr.fit(X_train, y_train)
svr_pred = svr.predict(X_test)
svr_mse = mean_squared_error(y_test, svr_pred)
svr_r2 = r2_score(y_test, svr_pred)
print("SVR Mean Squared Error:", svr_mse)
print("SVR R^2 Score:", svr_r2)
```

```
SVR Mean Squared Error: 595938.4511885272
SVR R^2 Score: 0.02984850925439042
```

```
from sklearn.model_selection import train_test_split

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

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso

models = [    LinearRegression(),    Ridge(alpha=0.5),    Lasso(alpha=0.1)]
```

```
from sklearn.metrics import mean_squared_error, r2_score

def evaluate(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return mse, r2
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'alpha': [0.1, 0.5, 1.0, 10.0]
}

grid_search = GridSearchCV(estimator=Ridge(), param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
```

```
mse, r2 = evaluate(best_model, X_test, y_test)
print('MSE:', mse)
print('R^2:', r2)
```

```
MSE: 45650.686461897545
R^2: 0.9435825667886149
```

Here are some guidelines for implementing machine learning in the Data Diary app:

Define the problem: Before implementing machine learning, it is essential to define the problem that needs to be solved. In this case, the problem is to predict the number of calories burned based on weight, height, age, and activity level.

Gather data: The quality of the machine learning model depends on the quality of the data used to train the model. The Data Diary app can automatically collect data from various sources, including the user's device and services. The app can also allow users to import data from public calendar feeds using iCal feeds.

Preprocess the data: Once the data is collected, it needs to be preprocessed before it can be used to train a machine learning model. This includes handling missing data, removing duplicates, and encoding categorical variables.

Perform exploratory data analysis (EDA): EDA is the process of analyzing and visualizing data to extract insights and identify patterns. This step is crucial to understand the data and select the appropriate machine learning algorithms.

Train the model: After preprocessing the data, the next step is to train a machine learning model. This involves selecting the appropriate algorithm, tuning hyperparameters, and evaluating the performance of the model.

Deploy the model: Once the model is trained, it can be deployed in the Data Diary app to make predictions on new data. It is essential to ensure that the model is integrated with the app securely and follows best practices for model deployment.

Monitor and improve the model: Machine learning models are not static and need to be monitored and improved continuously. This includes monitoring the performance of the model, collecting feedback from users, and retraining the model with new data.

In addition to these guidelines, here are some best practices for implementing machine learning in the Data Diary app:

Use appropriate algorithms: There are various machine learning algorithms available, and it is essential to select the appropriate algorithm based on the problem statement and the characteristics of the data.

Handle missing data: Missing data can significantly affect the performance of a machine learning model. It is essential to handle missing data appropriately by either removing the missing values or imputing them using appropriate techniques.

Regularize the model: Regularization is a technique used to prevent overfitting of the model. It is essential to regularize the model to ensure that it generalizes well on new data.
Evaluate the model: Evaluating the performance of a machine learning model is crucial to ensure that it meets the desired accuracy and performance criteria. It is essential to evaluate the model on a test set and use appropriate evaluation metrics.

Ensure model interpretability: Model interpretability is essential to understand the model's predictions and build trust with users. It is essential to use algorithms that are interpretable or use techniques such as SHAP values to explain the model's predictions.

Secure the model: Machine learning models can be vulnerable to attacks such as adversarial attacks and data poisoning attacks. It is essential to ensure that the model is secure and follows best practices for model security.

Follow ethical guidelines: Machine learning models can have ethical implications, and it is essential to follow ethical guidelines such as fairness, transparency, and accountability. It is essential to ensure that the model does not discriminate against certain groups and is transparent in its decision-making process.