code explanation buurahhhh

1. Importing Libraries:

At the start, we import libraries that help with handling data (like `pandas`), performing mathematical calculations (like `numpy`), and visualizing data (like `matplotlib` and `seaborn`). We also bring in machine learning algorithms and utilities for splitting data and evaluating models from `scikit-learn`.

2. Data Loading:

```
data = pd.read_csv(r"C:\Users\Kuro7\Documents\Medical diabities prediction\diabetes.csv")
```

This line loads the dataset, which contains information about different medical measurements related to diabetes.

3. Basic Data Exploration:

```
print(data.head())
print(data.describe())
```

We use these commands to look at the first few rows of the dataset and basic statistics, such as average and minimum values, to get a feel for the data.

4. Correlation and Visualization:

We plot a heatmap to check how different features (like glucose level, BMI, etc.) relate to each other and visualize the distributions using histograms and pie charts. These visualizations help to understand patterns in the data.

```
sns.heatmap(correlation_matrix, annot=True)
plt.show()

# Glucose level distribution
data['Glucose'].hist()
plt.show()

# Pie chart for diabetes outcomes
```

```
plt.pie(outcome_counts, ...)
plt.show()
```

These show how the different medical measurements relate to each other and the proportion of people with and without diabetes.

5. Cleaning Data (Handling Missing or Zero Values):

Some features like 'BloodPressure', 'SkinThickness', 'Insulin', and 'BMI' have values of zero, which are unrealistic. The code replaces these zeros with average values based on whether the patient has diabetes or not.

```
def replace_zero(df, field, target):
    mean_by_target = df.loc[df[field] != 0, ...].groupby(target).mean()
    df.loc[(df[field] == 0) & (df[target] == 0), field] =
mean_by_target.iloc[0][0]
    df.loc[(df[field] == 0) & (df[target] == 1), field] =
mean_by_target.iloc[1][0]
```

6. Outlier Detection:

Outliers are extreme values that could distort the results of the model. The code finds and removes them using the Interquartile Range (IQR) method.

```
def detect_outliers(df, nameOfFeature, drop=False):
    Q1 = valueOfFeature.quantile(0.25)
    Q3 = valueOfFeature.quantile(0.75)
    step = iqr(valueOfFeature) * 1.5
    outliers = df[(valueOfFeature < Q1 - step) | (valueOfFeature > Q3 +
step)].index.tolist()
```

This cleans up the data further by removing extreme values.

7. Splitting the Data:

We divide the dataset into two parts: one for training the model (`X_train` and `y_train`) and one for testing the model (`X_test` and `y_test`).

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

This makes sure the model is tested on data it hasn't seen during training.

8. Training Models:

The code uses multiple machine learning models, including Logistic Regression, K-Nearest Neighbors, Decision Trees, Random Forest, AdaBoost, and Naive Bayes. Each model is trained using the training data.

```
classifiers = {
    'Logistic Regression': LogisticRegression(random_state=SEED),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(random_state=SEED),
    'Random Forest': RandomForestClassifier(n_estimators=100,
random_state=SEED),
    'AdaBoost': AdaBoostClassifier(random_state=SEED),
    'Naive Bayes': GaussianNB()
}
```

These models are trained using:

```
def train_clf(clf, X_train, y_train):
    return clf.fit(X_train, y_train)
```

9. Evaluating Models:

After training, we test the models to see how well they predict diabetes on unseen data. We measure their performance using the F1 score (a metric that balances precision and recall) and accuracy.

```
def pred_clf(clf, features, target):
    y_pred = clf.predict(features)
    return f1_score(target, y_pred, pos_label=1)

def accu_clf(clf, features, target):
    y_pred = clf.predict(features)
    return accuracy_score(target, y_pred)
```

These functions give us a clear idea of how each model performs.

10. Visualizing Results:

To easily compare the models, the code creates bar charts showing the F1 scores and accuracies for each model.

```
plt.bar(index, train_f1_scores, ...)
plt.bar(index + bar_width, test_f1_scores, ...)
plt.show()

# Plot accuracy
sns.barplot(x=clf_names, y=test_accuracies, ...)
plt.show()
```

This makes it easier to visualize which model works best.

11. **Feature Importance** (Optional):

Random Forest can tell us which features are the most important for making predictions. We plot the importance of each feature.

```
feature_importances = rf_model.feature_importances_

sns.barplot(x=feature_importances, y=X.columns, ...)
plt.show()
```

Summary:

In short, this code:

1. Loads and cleans the data.
2. Visualizes relationships and patterns in the data.
3. Trains several machine learning models to predict whether someone has diabetes.
4. Evaluates the models using F1 score and accuracy.
5. Visualizes the performance of different models and the importance of features.

This pipeline can be used as a template for other health-related predictions as well, simply by swapping out the dataset and making some adjustments!