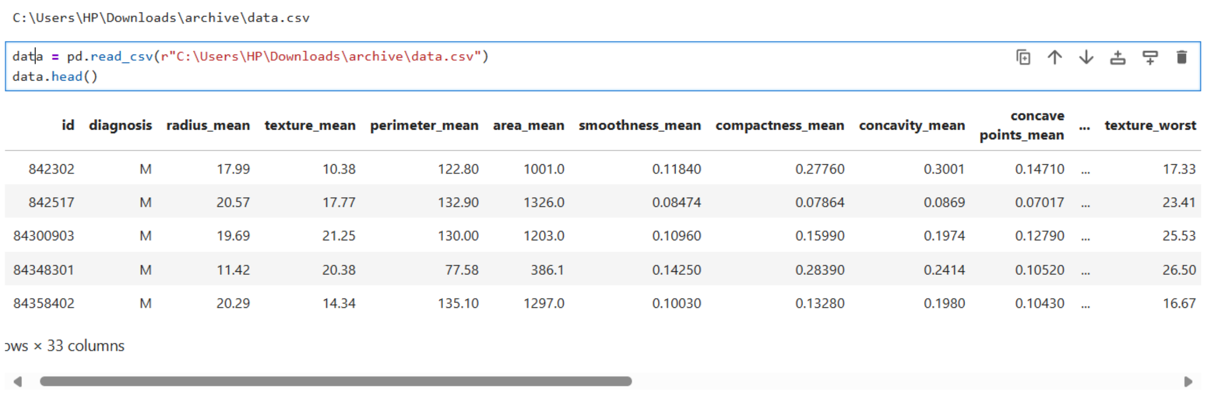
Workflow

The analysis follows these key steps:

1. Data Preprocessing

First, we separate the dataset into our predictors (the cell measurements), conventionally named X, and the target variable (the diagnosis), named y.



```
C:\Users\HP\Downloads\archive\data.csv
```

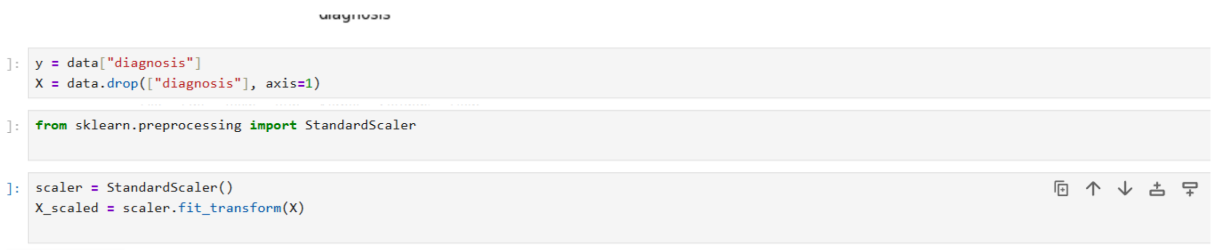
```python
data = pd.read_csv(r"C:\Users\HP\Downloads\archive\data.csv")
data.head()
```

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst
842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33
842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41
84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53
84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50
84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67

ows × 33 columns

2. Normalization

Before training, the predictor data X is normalized using Scikit-Learn's StandardScaler. This step is crucial because it scales all features to a similar range, preventing variables with larger units from disproportionately influencing the model. Normalization improves model performance and stability.



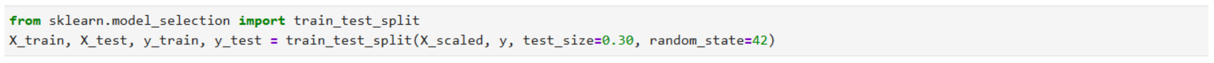
```python
y = data["diagnosis"]
X = data.drop(["diagnosis"], axis=1)
```

```python
from sklearn.preprocessing import StandardScaler
```

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3. Train-Test Split

The normalized dataset is split into a training set and a testing set using the train_test_split function.



```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.30, random_state=42)
```

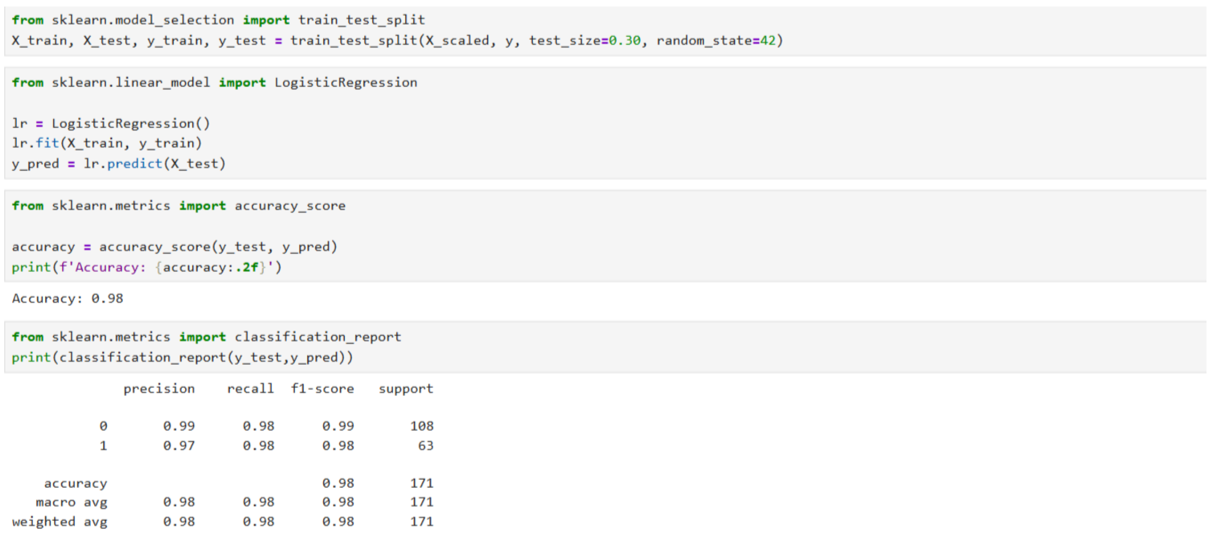
X_train & y_train: Data used to train the logistic regression model.

X_test & y_test: Unseen data used to evaluate the model's performance.

We use a random_state (e.g., 42) to ensure that the split is reproducible.

4. Model Evaluation

After training the model on the X_train data, we evaluate its performance on the unseen X_test data. The primary metric for this project is accuracy, but we also generate a full classification_report to assess other important metrics like precision, recall, and F1-score.



```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.30, random_state=42)
```

```python
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
```

```python
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.98
```

```python
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       108
           1       0.97      0.98      0.98        63

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

Results

The trained logistic regression model achieved a final accuracy of 0.98 (98%) on the test set. This excellent performance indicates that the model is highly effective at making correct predictions on new, unseen data.

Potential Applications

A trained model like this has significant real-world potential beyond academic analysis. It can be a powerful tool for medical professionals.

Since the model can be easily called within a Python function, it can be integrated into a larger software application. For example, it could be deployed on a server using a framework like Flask to power a front-end application. A doctor could use this application to input cell measurements and receive an instant diagnosis prediction from the model.

A more advanced and realistic application could involve connecting this model's backend to a laboratory machine. The machine could automatically sample tissue, measure the cells, and use the model to perform an initial diagnosis, significantly speeding up the analytical process. By leveraging

Python, the API for such an application would be minimalist, demonstrating how a straightforward logistic regression model can be used to build life-saving tools.