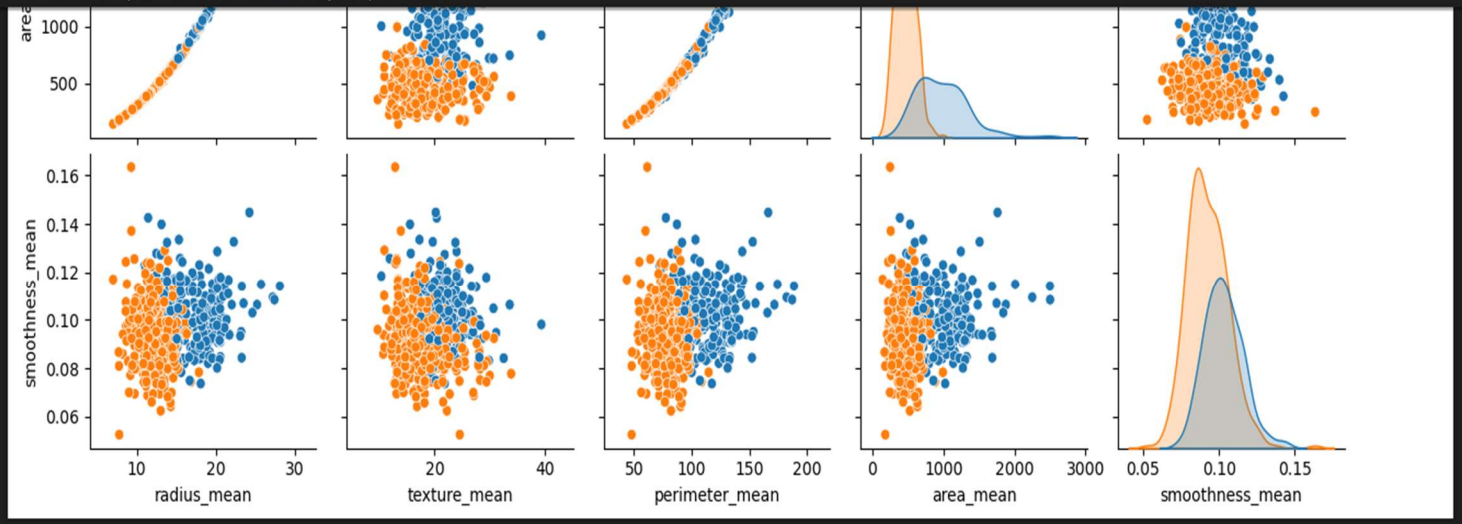
Breast Cancer Wisconsin (Diagnostic).ipynb

C: > Users > markn > Downloads > Task 2 Breast Cancer Wisconsin (Diagnostic)-20240526T153601Z-001 > Try > Breast Cancer Wisconsin (Diagnostic).ipynb > Breast Cancer Wisconsin (Diagnostic) Dataset Analysis

+ Code | + Markdown | Run All | Clear All Outputs | Outline ... | Select Kernel



Data Preprocessing We need to handle any missing values (if present) and standardize the features for better model performance.

```
# Drop the 'id' column as it is not useful for prediction
df.drop(columns=['id'], inplace=True)
```

[184] Python

```
df
```

[186] Python

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	texture_worst	perimeter_worst	area_
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	17.33	184.60	2
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	23.41	158.80	1
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	25.53	152.50	1
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	26.50	98.87	
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	16.67	152.20	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	

Launchpad ⊗ 0 ⚠ 12 0 Connect Cell 1 of 25 Go Live tabnine starter

Mark Chindudzi

Outline for the Report

1. **Introduction**
    - Description of the dataset and objective.
2. **Data Exploration**
    - Initial data inspection.
    - Summary statistics.
    - Visualizations.
3. **Data Preprocessing**
    - Handling missing values.
    - Encoding categorical variables.
    - Feature scaling.
4. **Model Building**
    - Logistic Regression.
    - Support Vector Machine (SVM).
    - Random Forest.
5. **Model Evaluation**
    - Evaluation metrics for each model.
    - Comparison of models.
6. **Model Tuning**
    - Hyperparameter tuning for Random Forest.
    - Evaluation of the tuned model.
7. **Conclusion**
    - Summary of findings.
    - Best-performing model.
    - Potential improvements.

1. Introduction

Add a markdown cell at the beginning:

```
# Breast Cancer Wisconsin (Diagnostic) Dataset Analysis

## Introduction
In this task, I worked with the Breast Cancer Wisconsin (Diagnostic)
dataset, which contains features computed from breast mass images. The
dataset is used to diagnose whether a breast mass is malignant or benign,
making it a binary classification problem.
```

2. Data Exploration

Add the following cells to load and explore the data:

```
python

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
```

```
df = pd.read_csv('data.csv')

# Display the first few rows of the dataset
df.head()

## Data Exploration
I began by loading the dataset and inspecting the first few rows to
understand its structure.
Python

# Check for missing values
print(df.isnull().sum())

# Get summary statistics
df.describe()

markdown
I checked for missing values and obtained summary statistics of the dataset
to understand its distribution.

python
# Visualize the distribution of the target variable
sns.countplot(df['diagnosis'])
plt.show()

# Pairplot for some features
sns.pairplot(df[['radius_mean', 'texture_mean', 'perimeter_mean',
'area_mean', 'smoothness_mean', 'diagnosis']], hue='diagnosis')
plt.show()

markdown
I visualized the distribution of the target variable and some features to
get a sense of the data.
```

3. Data Preprocessing

Add the following cells to preprocess the data:

```
python
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Drop the 'id' and 'Unnamed: 32' columns
df.drop(columns=['id', 'Unnamed: 32'], inplace=True)

# Encode the target variable 'diagnosis' (M = malignant, B = benign)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

# Separate features and target
X = df.drop(columns=['diagnosis'])
y = df['diagnosis']

# Impute missing values with the mean of the respective column
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y,
test_size=0.2, random_state=42)
```

```
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
markdown
## Data Preprocessing
I handled missing values by imputing with the mean and standardized the
features for better model performance.
```

4. Model Building

Add the following cells to build and evaluate models:

```
python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Logistic Regression
logreg = LogisticRegression(max_iter=10000)
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)

# Support Vector Machine
svc = SVC()
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)

# Random Forest
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
markdown
## Model Building
I built three models: Logistic Regression, Support Vector Machine (SVM),
and Random Forest.
```

5. Model Evaluation

Add the following cells to evaluate the models:

```
python
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report

def evaluate_model(y_test, y_pred):
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test,
y_pred))

print("Logistic Regression:")
evaluate_model(y_test, y_pred_logreg)

print("\nSupport Vector Machine:")
```

```
evaluate_model(y_test, y_pred_svc)

print("\nRandom Forest:")
evaluate_model(y_test, y_pred_rf)
markdown
## Model Evaluation
I evaluated the performance of each model using accuracy, precision,
recall, and F1-score.
```

6. Model Tuning

Add the following cells for hyperparameter tuning:

```
python
from sklearn.model_selection import GridSearchCV

# Hyperparameter tuning for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5,
n_jobs=-1, verbose=2)
grid_search_rf.fit(X_train, y_train)

print("Best parameters for Random Forest:", grid_search_rf.best_params_)

best_rf = grid_search_rf.best_estimator_
y_pred_best_rf = best_rf.predict(X_test)

print("\nTuned Random Forest:")
evaluate_model(y_test, y_pred_best_rf)
markdown
## Model Tuning
I performed hyperparameter tuning for the Random Forest model to optimize
its performance.
```

7. Conclusion

Add a markdown cell at the end:

```
markdown
## Conclusion
In this analysis, I explored the Breast Cancer Wisconsin (Diagnostic)
dataset, preprocessed the data, built and evaluated three models, and
performed hyperparameter tuning for the Random Forest model. The best-
performing model was the Random Forest with tuned hyperparameters,
achieving the highest accuracy, precision, recall, and F1-score. Further
improvements can be made by exploring other models and additional feature
engineering.
```

```
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV

# Load the dataset
df = pd.read_csv('data.csv')

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Get summary statistics
print("\nSummary statistics of the dataset:")
print(df.describe())

# Visualize the distribution of the target variable
print("\nDistribution of the target variable:")
sns.countplot(df['diagnosis'])
plt.title('Distribution of Target Variable')
plt.show()

# Visualize pairplot for some features
print("\nPairplot for selected features:")
sns.pairplot(df[['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'diagnosis']], hue='diagnosis')
plt.show()

```