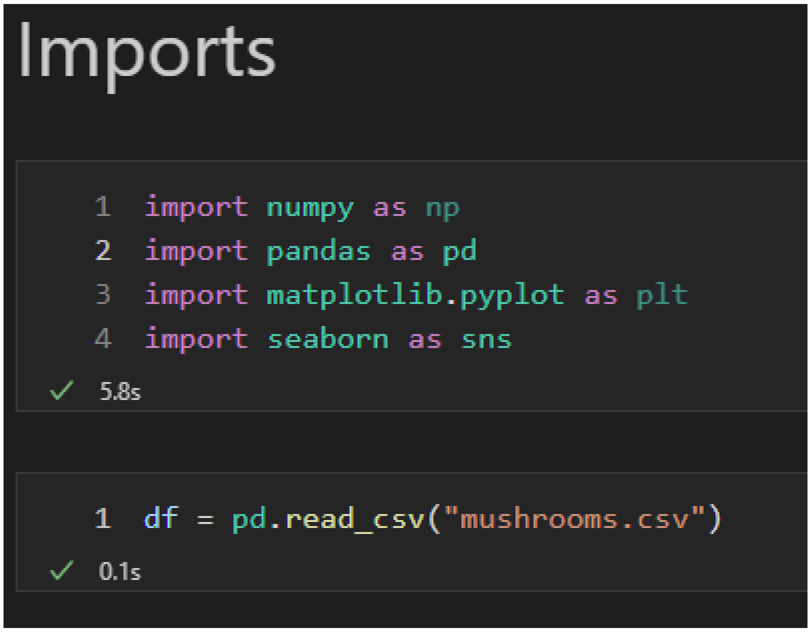
🌴

Boosting Methods

▼ ADA Mushroom



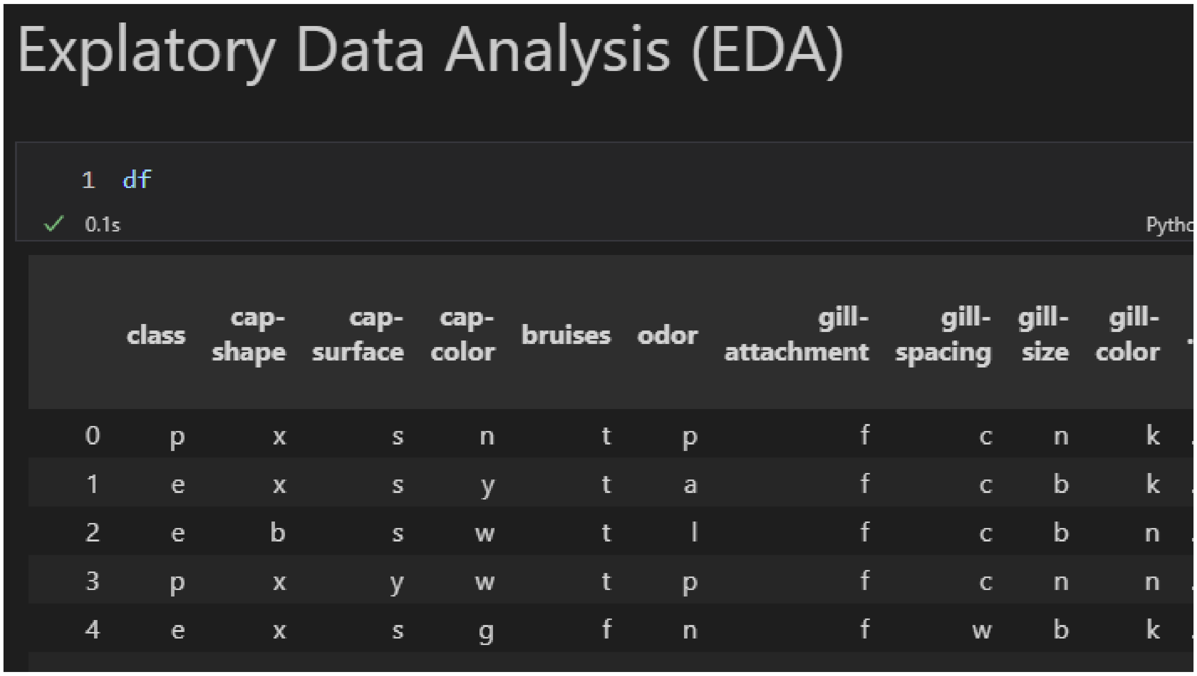
Imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

✓ 5.8s

```
df = pd.read_csv("mushrooms.csv")
```

✓ 0.1s

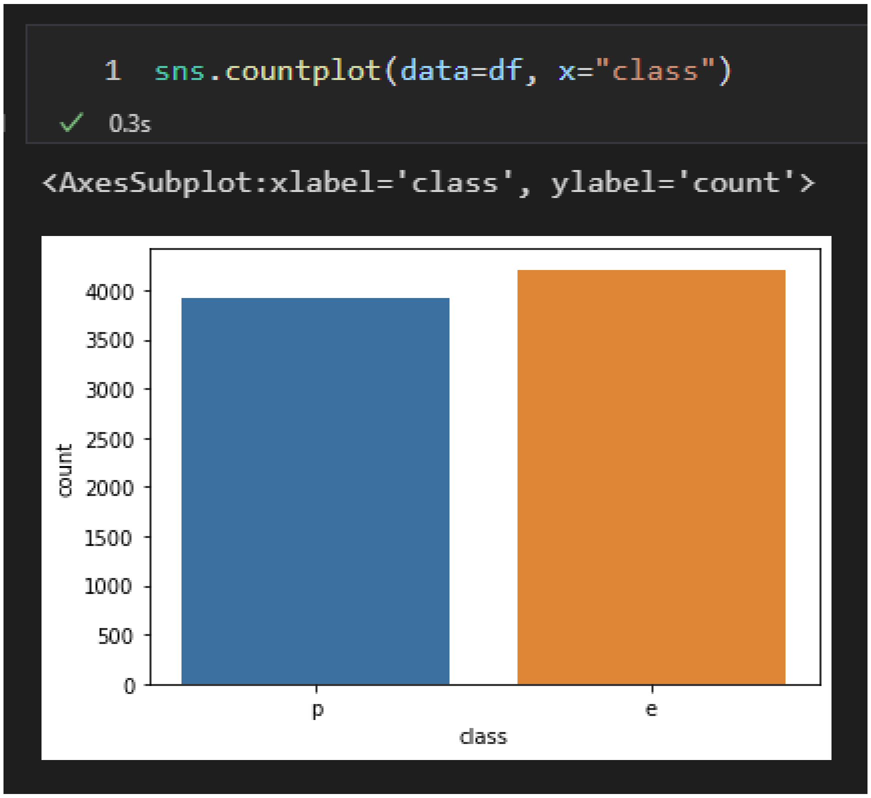


Explatory Data Analysis (EDA)

```
df
```

✓ 0.1s

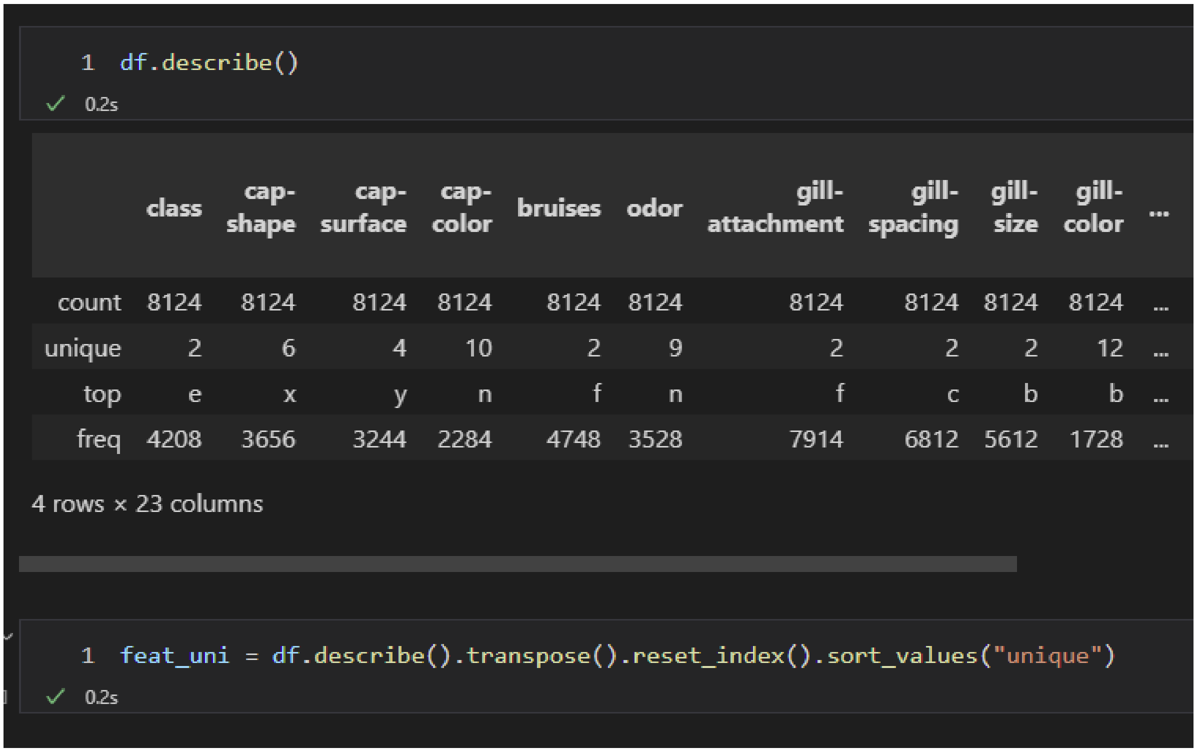
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color
0	p	x	s	n	t	p	f	c	n	k
1	e	x	s	y	t	a	f	c	b	k
2	e	b	s	w	t	l	f	c	b	n
3	p	x	y	w	t	p	f	c	n	n
4	e	x	s	g	f	n	f	w	b	k



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

✓ 0.3s

```
<AxesSubplot:xlabel='class', ylabel='count'>
```



```python
df.describe()
```

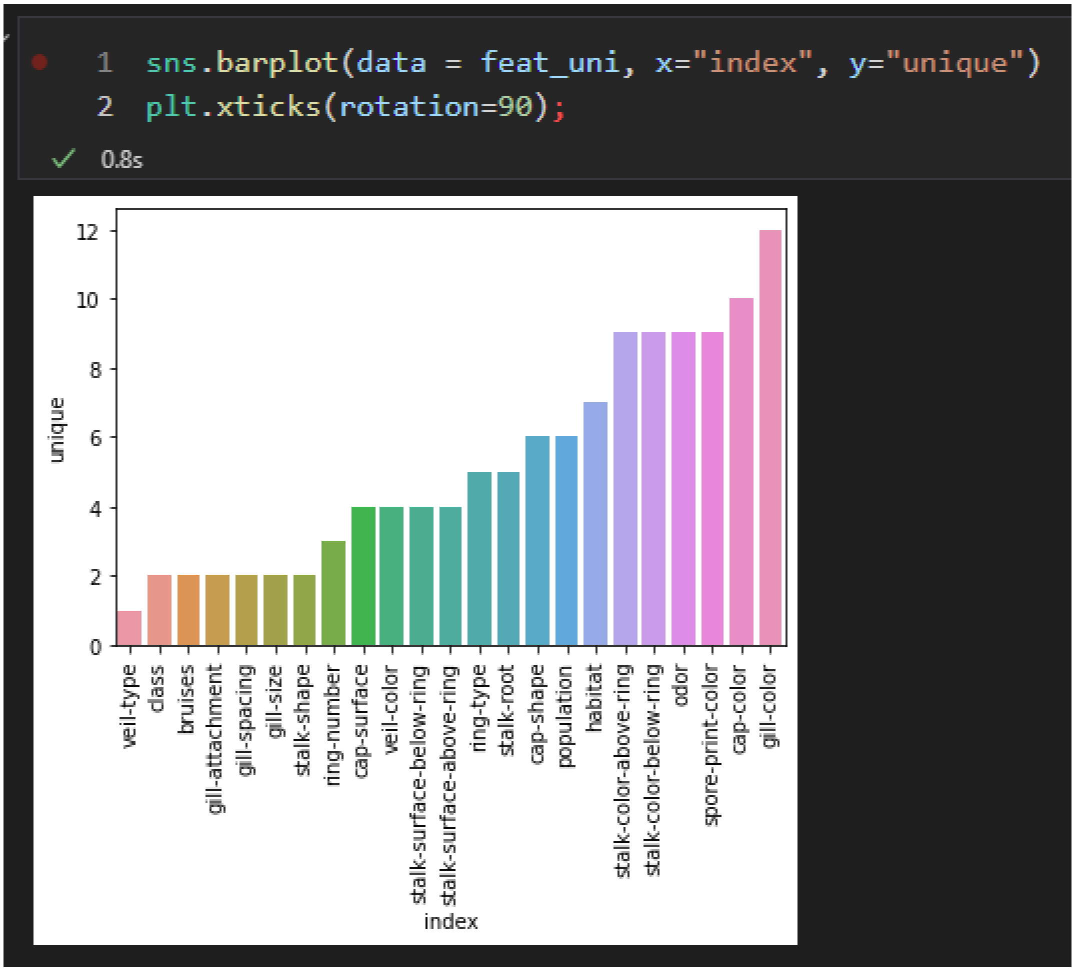
✓ 0.2s

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	...
unique	2	6	4	10	2	9	2	2	2	12	...
top	e	x	y	n	f	n	f	c	b	b	...
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	...

4 rows × 23 columns

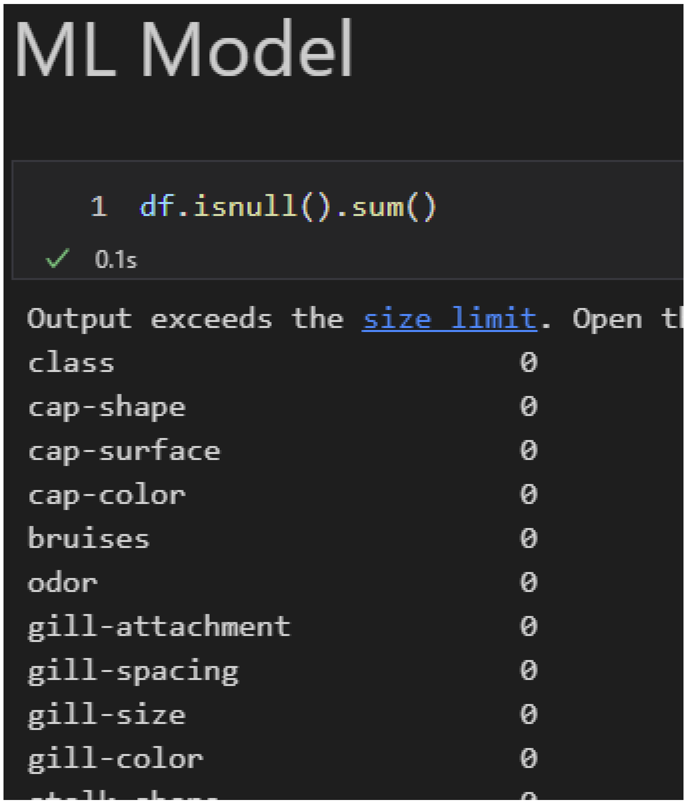
```python
feat_uni = df.describe().transpose().reset_index().sort_values("unique")
```

✓ 0.2s



```
sns.barplot(data = feat_uni, x="index", y="unique")
plt.xticks(rotation=90);
```

0.8s

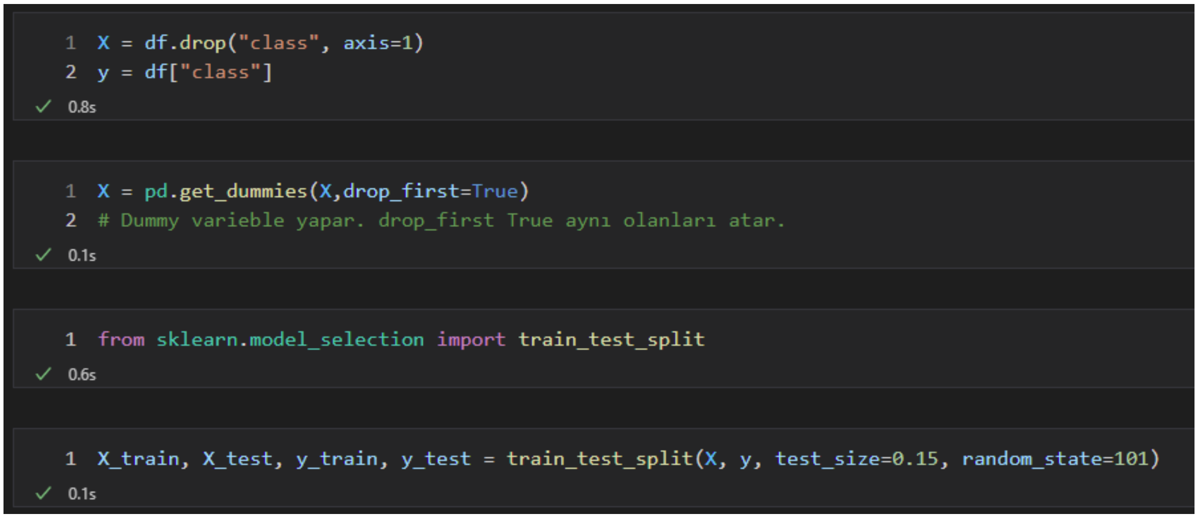


ML Model

```python
df.isnull().sum()
```

✓ 0.1s

```
Output exceeds the size limit. Open tl
class                        0
cap-shape                    0
cap-surface                  0
cap-color                    0
bruises                      0
odor                         0
gill-attachment              0
gill-spacing                 0
gill-size                    0
gill-color                   0
stalk-shape                  0
```



```python
X = df.drop("class", axis=1)
y = df["class"]
```

✓ 0.8s

```python
X = pd.get_dummies(X,drop_first=True)
# Dummy varieble yapar. drop_first True aynı olanları atar.
```

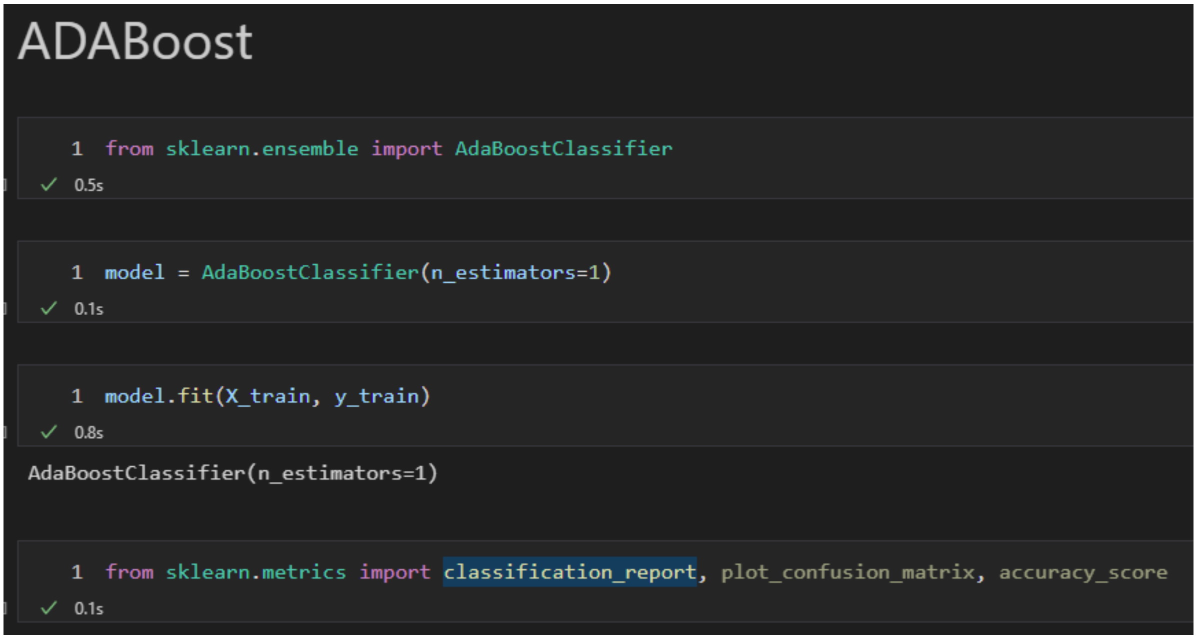
✓ 0.1s

```python
from sklearn.model_selection import train_test_split
```

✓ 0.6s

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=101)
```

✓ 0.1s



ADABoost

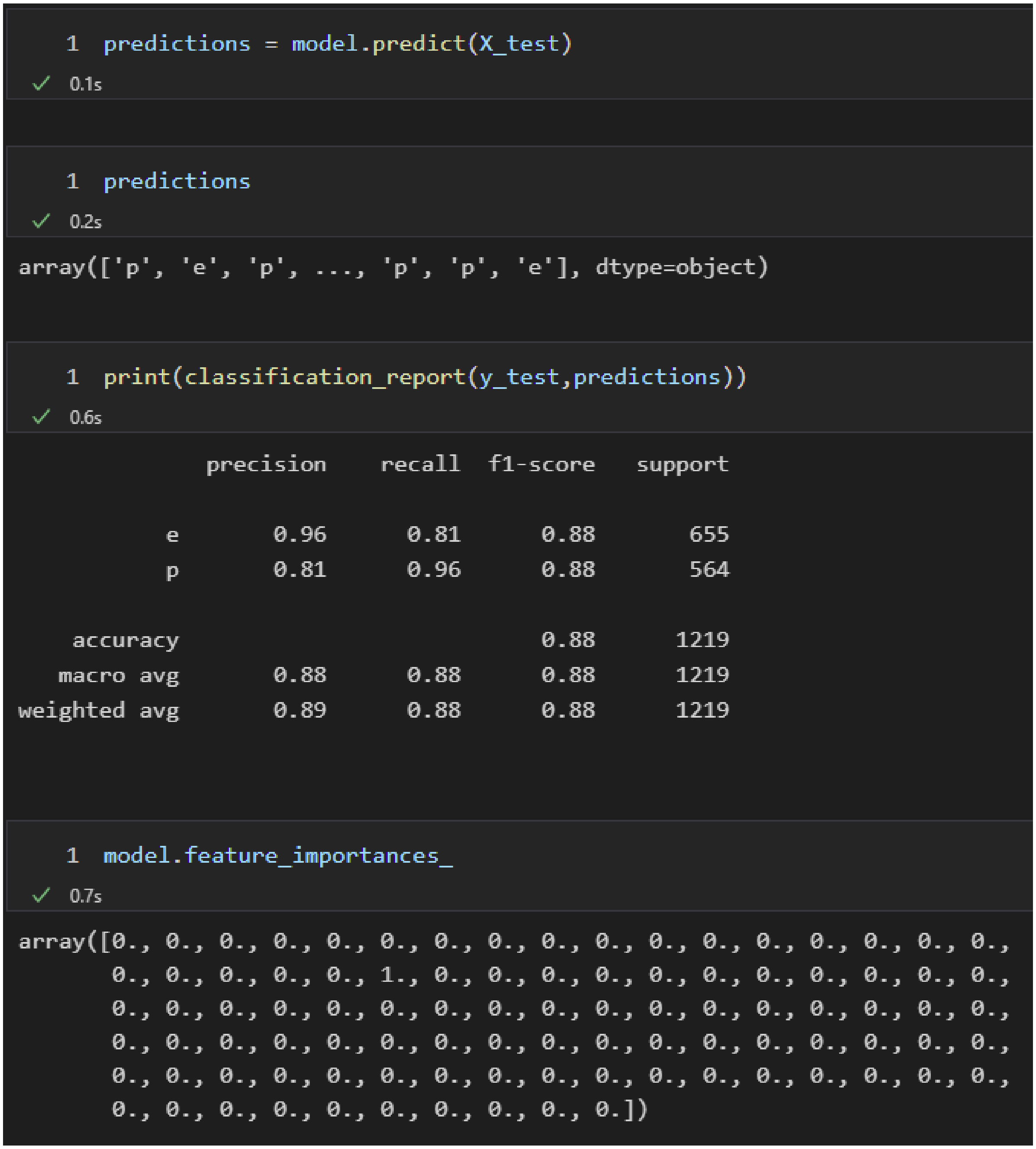
```python
from sklearn.ensemble import AdaBoostClassifier
```

```python
model = AdaBoostClassifier(n_estimators=1)
```

```python
model.fit(X_train, y_train)
```

```
AdaBoostClassifier(n_estimators=1)
```

```python
from sklearn.metrics import classification_report, plot_confusion_matrix, accuracy_score
```



```
predictions = model.predict(X_test)
```

```
predictions
```

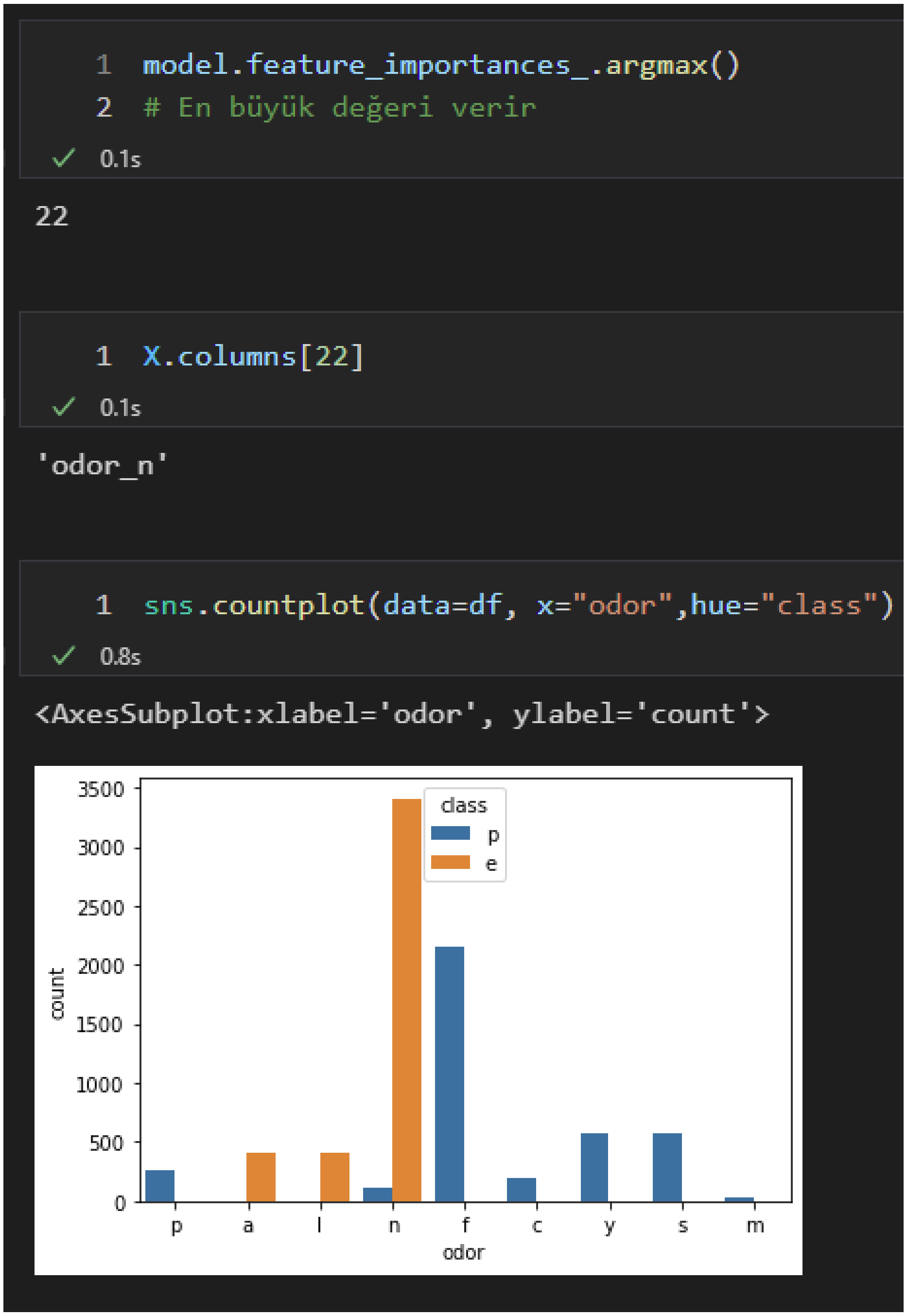
```
array(['p', 'e', 'p', ..., 'p', 'p', 'e'], dtype=object)
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           e       0.96      0.81      0.88       655
           p       0.81      0.96      0.88       564

    accuracy                           0.88      1219
   macro avg       0.88      0.88      0.88      1219
weighted avg       0.89      0.88      0.88      1219
```

```
model.feature_importances_
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```



```
model.feature_importances_.argmax()
# En büyük değeri verir
```

✓ 0.1s

```
22
```

```
X.columns[22]
```

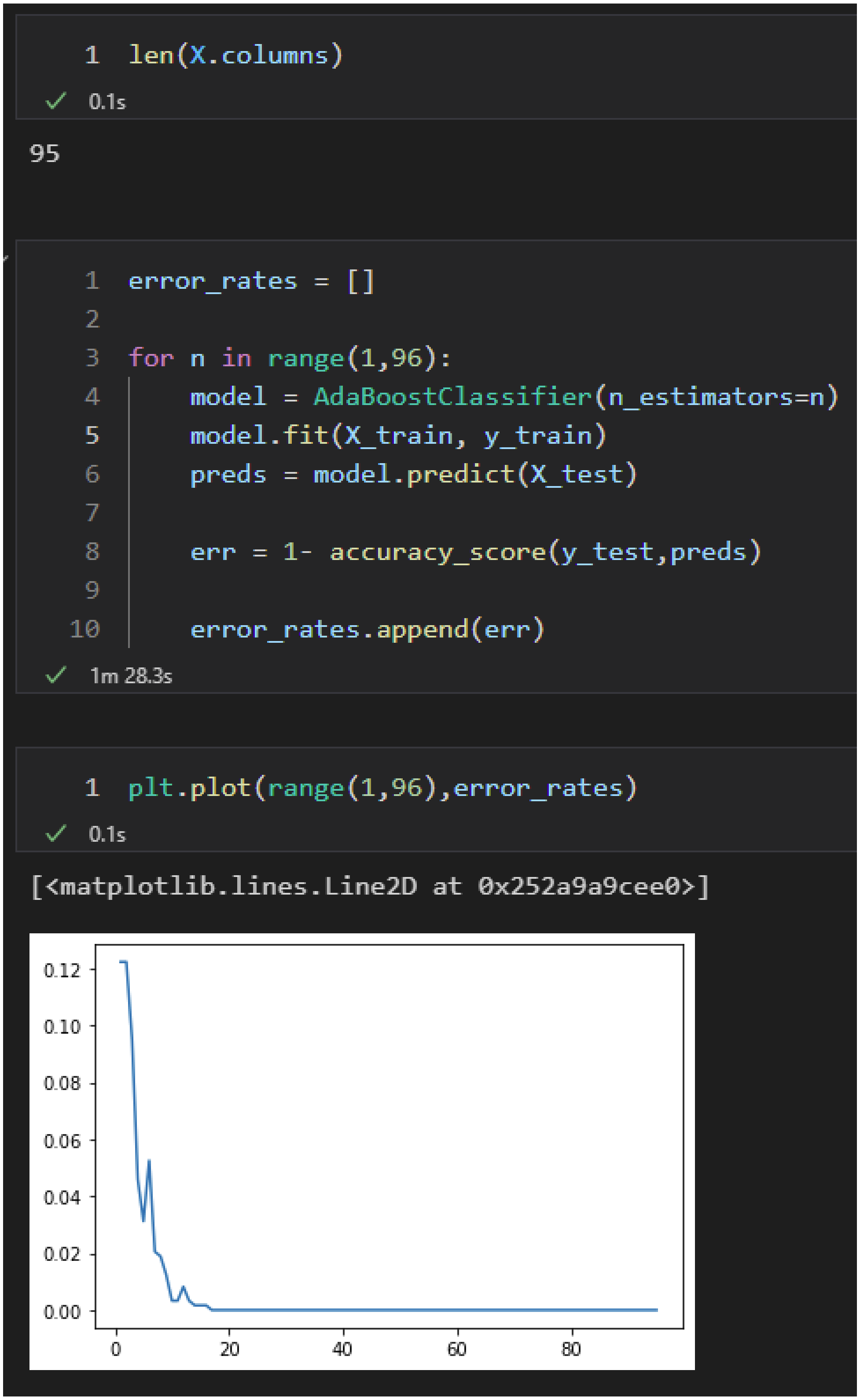
✓ 0.1s

```
'odor_n'
```

```
sns.countplot(data=df, x="odor",hue="class")
```

✓ 0.8s

```
<AxesSubplot:xlabel='odor', ylabel='count'>
```



```
len(X.columns)
```

✓ 0.1s

```
95
```

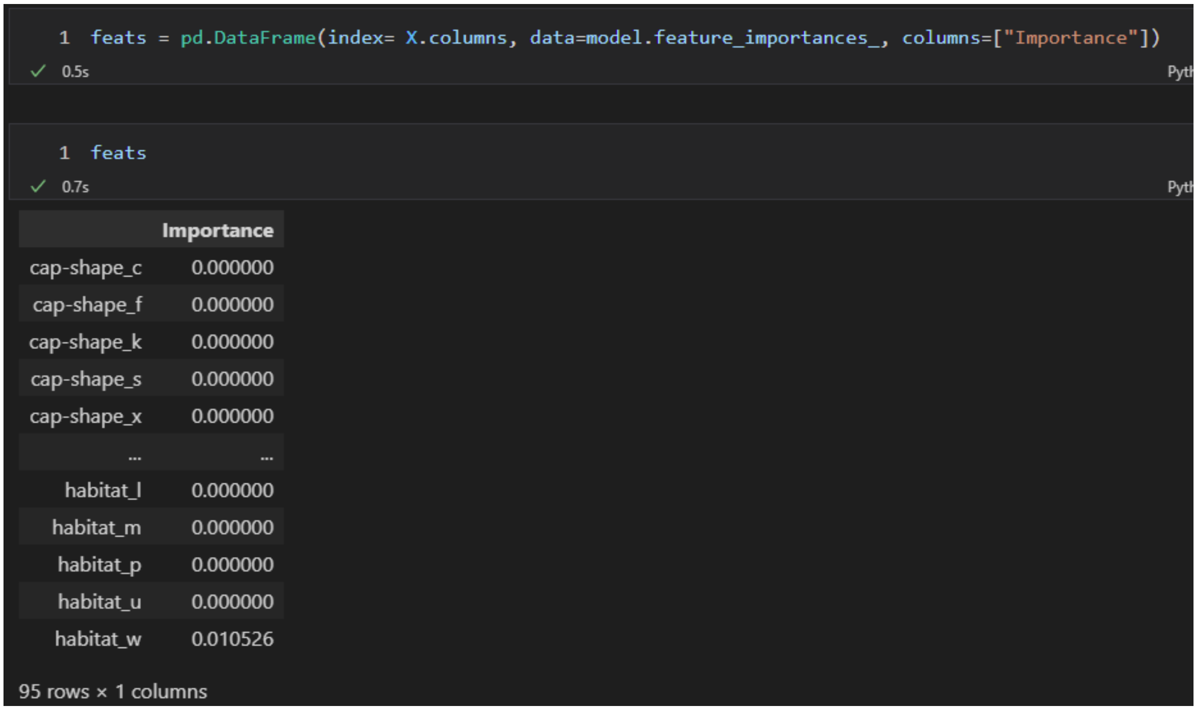
```
error_rates = []

for n in range(1,96):
    model = AdaBoostClassifier(n_estimators=n)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    err = 1- accuracy_score(y_test,preds)

    error_rates.append(err)
```

✓ 1m 28.3s

```
plt.plot(range(1,96),error_rates)
```

✓ 0.1s

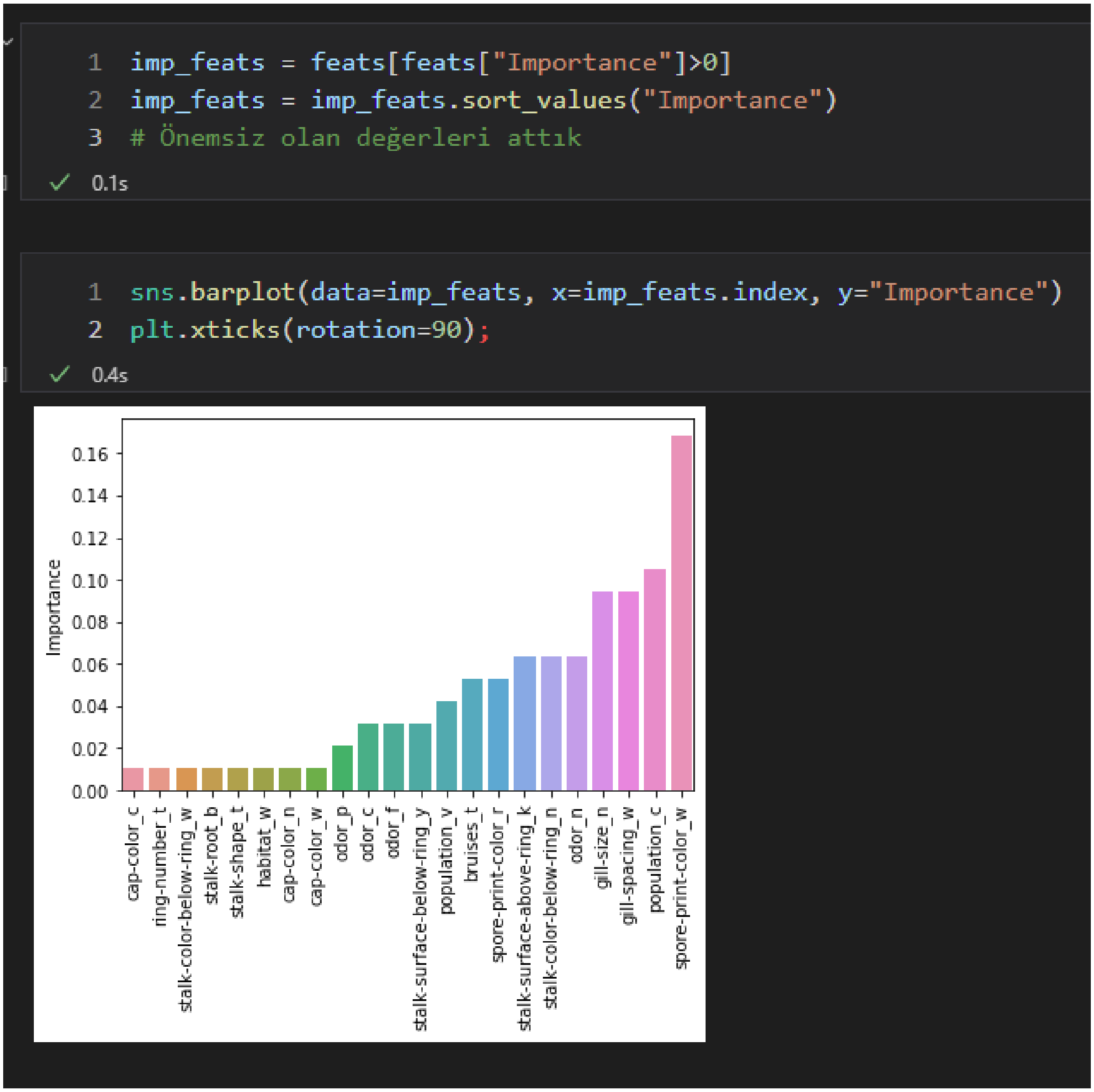
```
[<matplotlib.lines.Line2D at 0x252a9a9cee0>]
```



```python
feats = pd.DataFrame(index= X.columns, data=model.feature_importances_, columns=["Importance"])
```

```python
feats
```

```
              Importance
 cap-shape_c    0.000000
 cap-shape_f    0.000000
 cap-shape_k    0.000000
 cap-shape_s    0.000000
 cap-shape_x    0.000000
         ...         ...
   habitat_l    0.000000
   habitat_m    0.000000
   habitat_p    0.000000
   habitat_u    0.000000
   habitat_w    0.010526

95 rows × 1 columns
```

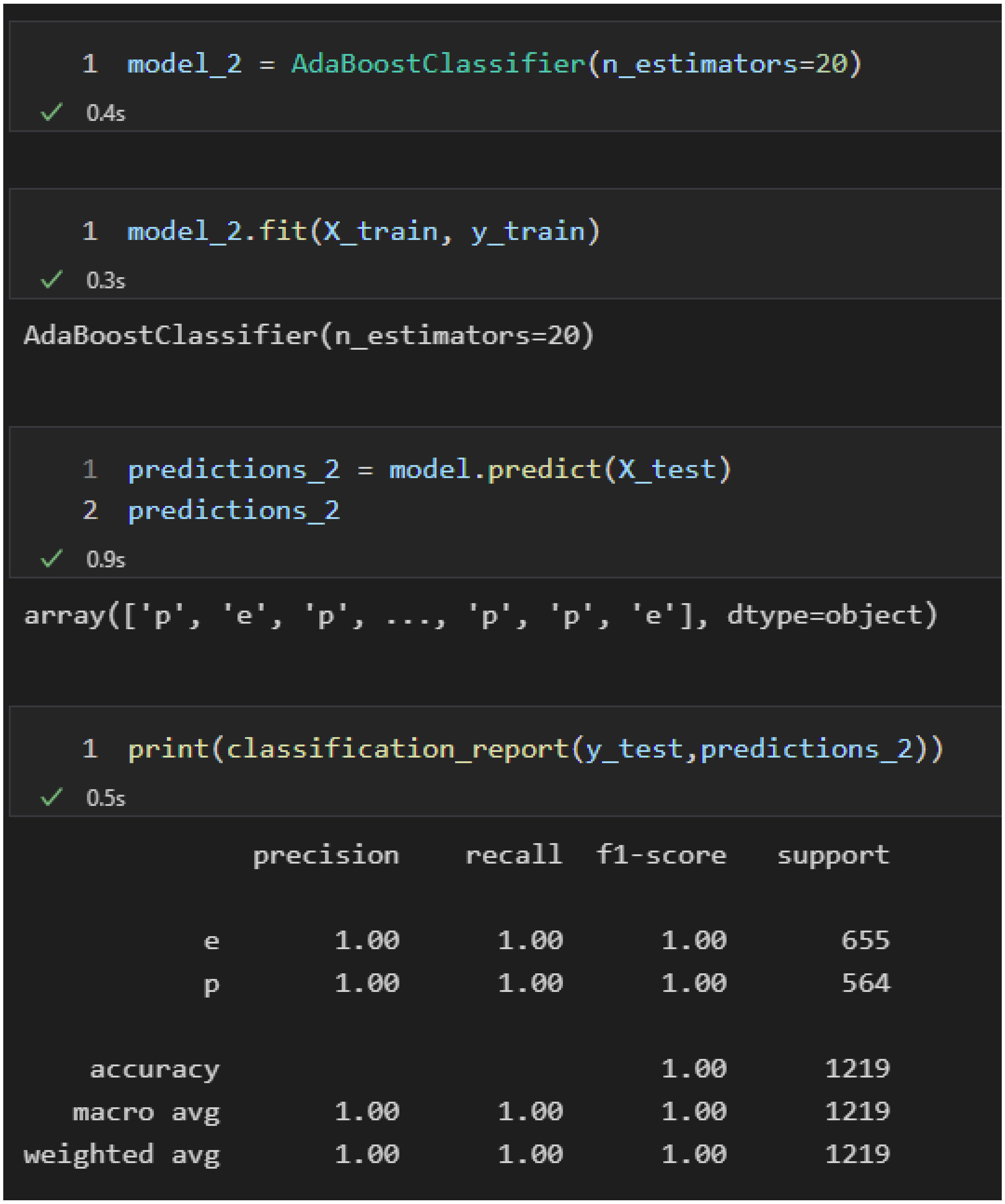


```
imp_feats = feats[feats["Importance"]>0]
imp_feats = imp_feats.sort_values("Importance")
# Önemsiz olan değerleri attık
```

✓ 0.1s

```
sns.barplot(data=imp_feats, x=imp_feats.index, y="Importance")
plt.xticks(rotation=90);
```

✓ 0.4s



```
model_2 = AdaBoostClassifier(n_estimators=20)
```

✓ 0.4s

```
model_2.fit(X_train, y_train)
```

✓ 0.3s

```
AdaBoostClassifier(n_estimators=20)
```

```
predictions_2 = model.predict(X_test)
predictions_2
```

✓ 0.9s

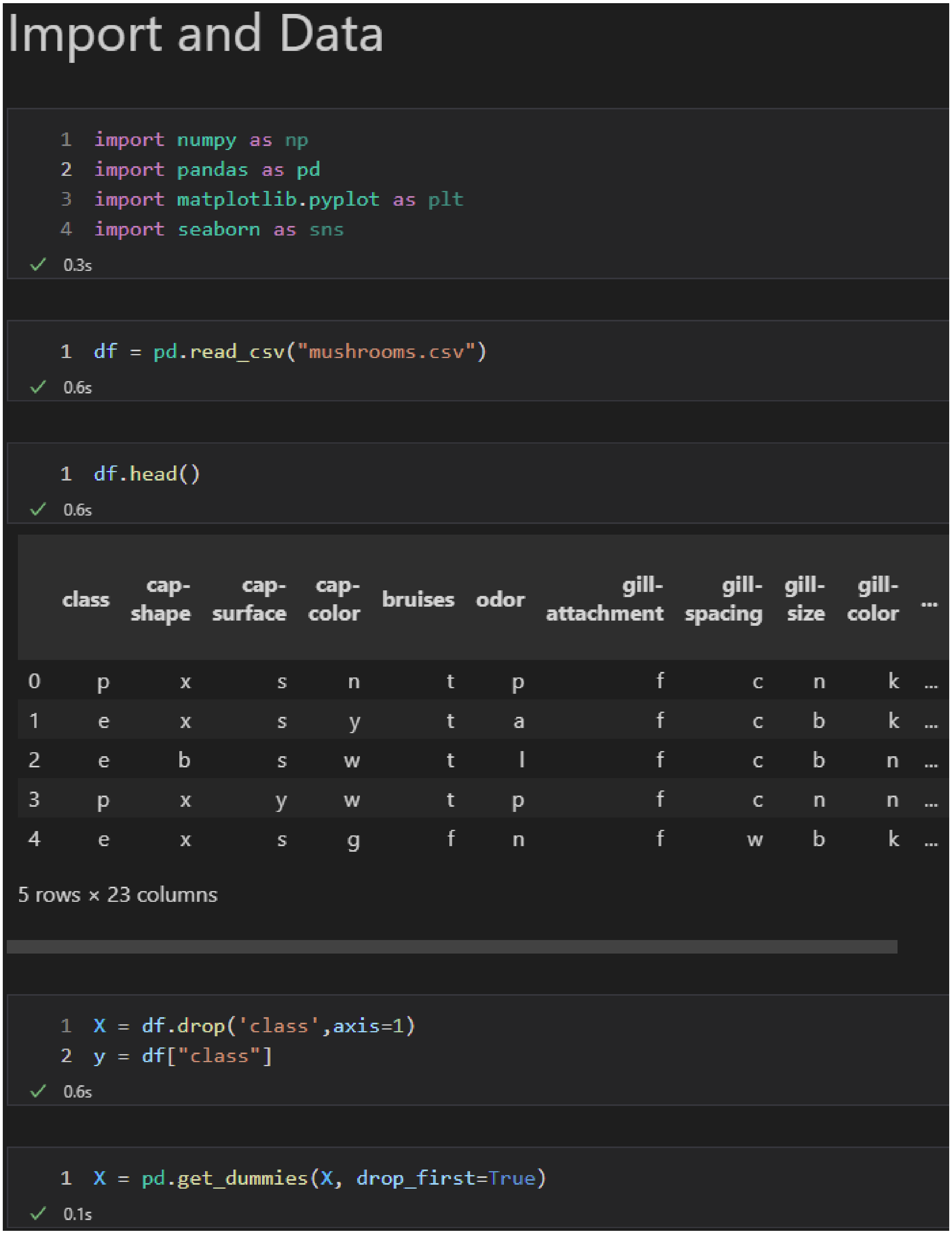
```
array(['p', 'e', 'p', ..., 'p', 'p', 'e'], dtype=object)
```

```
print(classification_report(y_test,predictions_2))
```

✓ 0.5s

```
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       655
           p       1.00      1.00      1.00       564

    accuracy                           1.00      1219
   macro avg       1.00      1.00      1.00      1219
weighted avg       1.00      1.00      1.00      1219
```

▼ Gradient Boosting Mushroom



Import and Data

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

✓ 0.3s

```python
df = pd.read_csv("mushrooms.csv")
```

✓ 0.6s

```python
df.head()
```

✓ 0.6s

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...
0	p	x	s	n	t	p	f	c	n	k	...
1	e	x	s	y	t	a	f	c	b	k	...
2	e	b	s	w	t	l	f	c	b	n	...
3	p	x	y	w	t	p	f	c	n	n	...
4	e	x	s	g	f	n	f	w	b	k	...

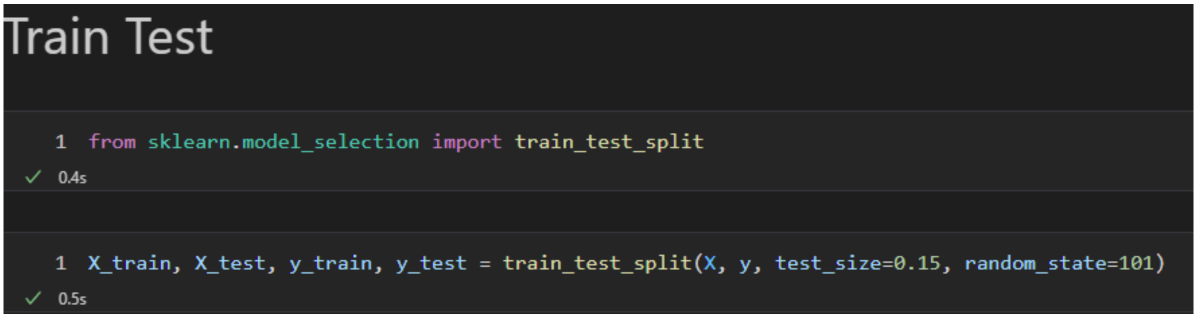
5 rows × 23 columns

```python
X = df.drop('class',axis=1)
y = df["class"]
```

✓ 0.6s

```python
X = pd.get_dummies(X, drop_first=True)
```

✓ 0.1s



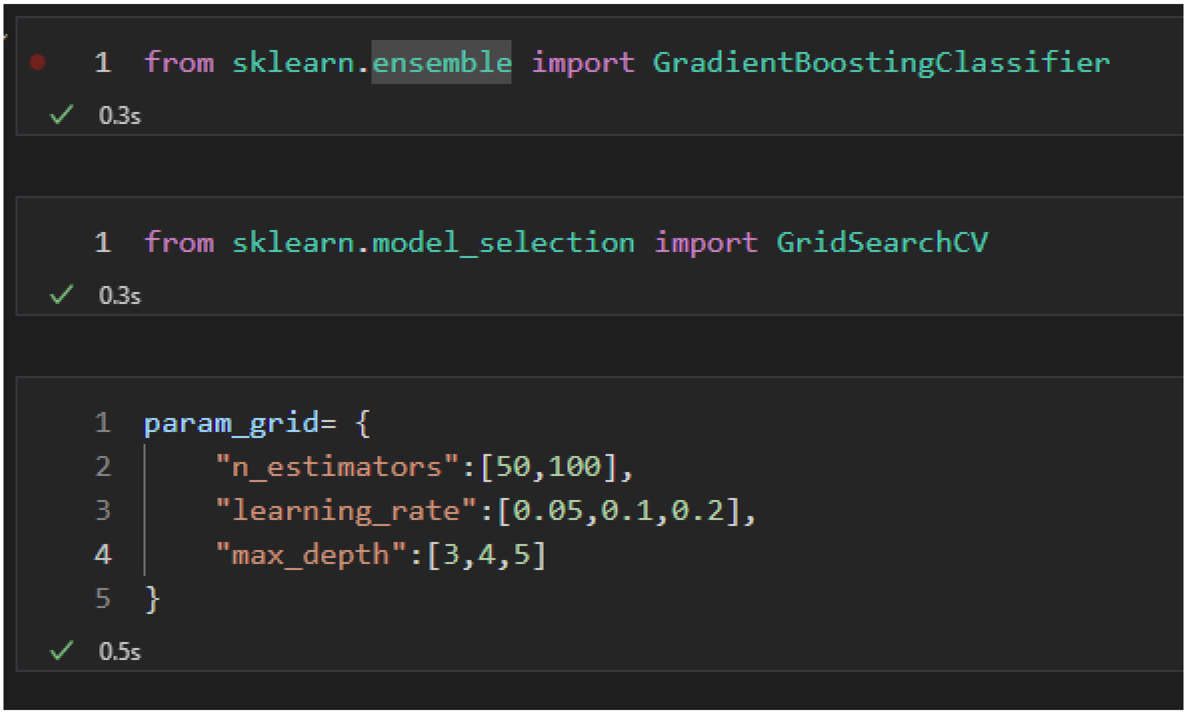
Train Test

```python
from sklearn.model_selection import train_test_split
```

✓ 0.4s

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=101)
```

✓ 0.5s



```python
from sklearn.ensemble import GradientBoostingClassifier
```

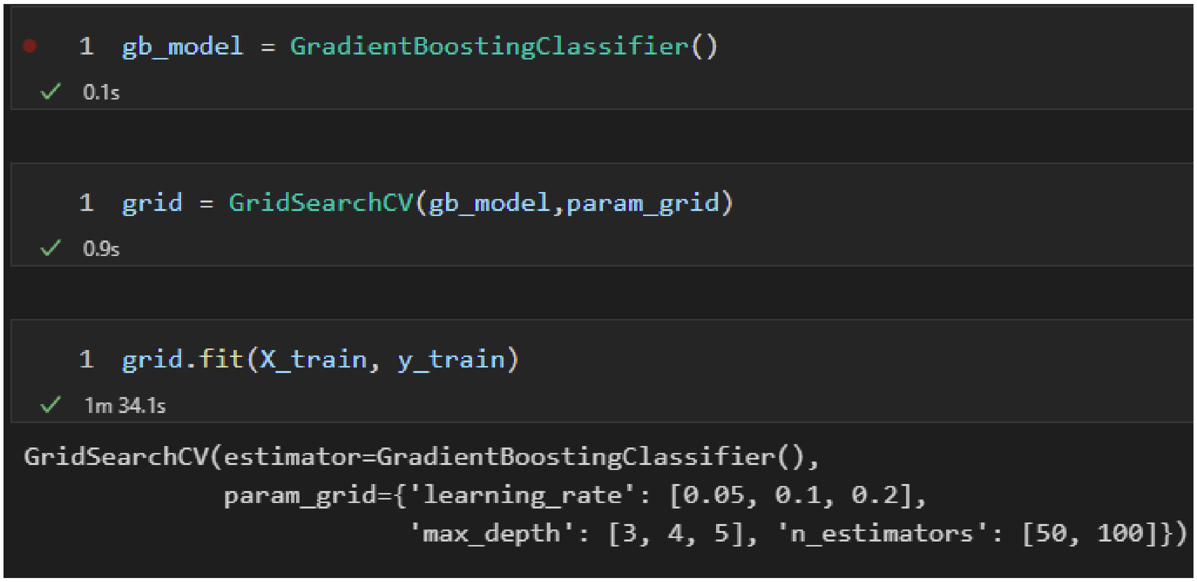
✓ 0.3s

```python
from sklearn.model_selection import GridSearchCV
```

✓ 0.3s

```python
param_grid= {
    "n_estimators":[50,100],
    "learning_rate":[0.05,0.1,0.2],
    "max_depth":[3,4,5]
}
```

✓ 0.5s



```python
gb_model = GradientBoostingClassifier()
```

✓ 0.1s

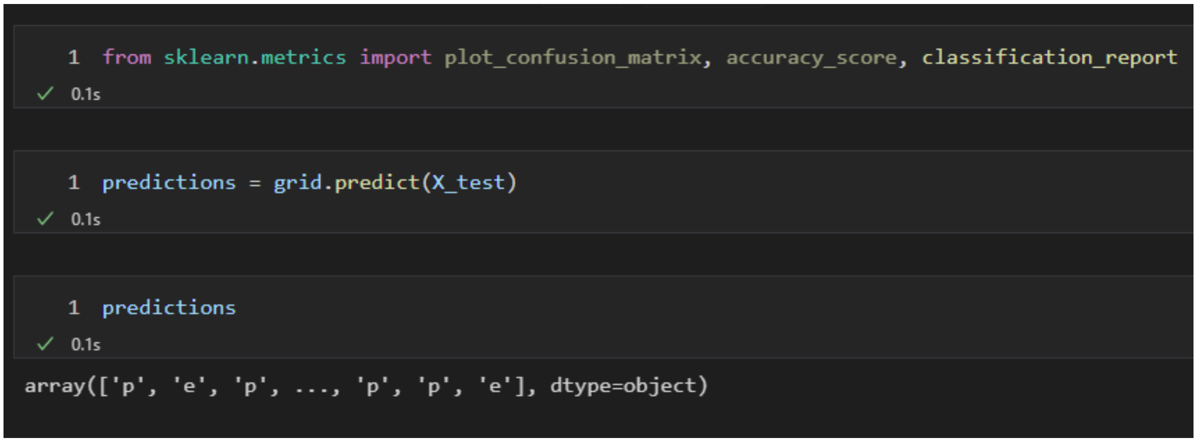
```python
grid = GridSearchCV(gb_model,param_grid)
```

✓ 0.9s

```python
grid.fit(X_train, y_train)
```

✓ 1m 34.1s

```
GridSearchCV(estimator=GradientBoostingClassifier(),
             param_grid={'learning_rate': [0.05, 0.1, 0.2],
                         'max_depth': [3, 4, 5], 'n_estimators': [50, 100]})
```



```python
from sklearn.metrics import plot_confusion_matrix, accuracy_score, classification_report
```

✓ 0.1s

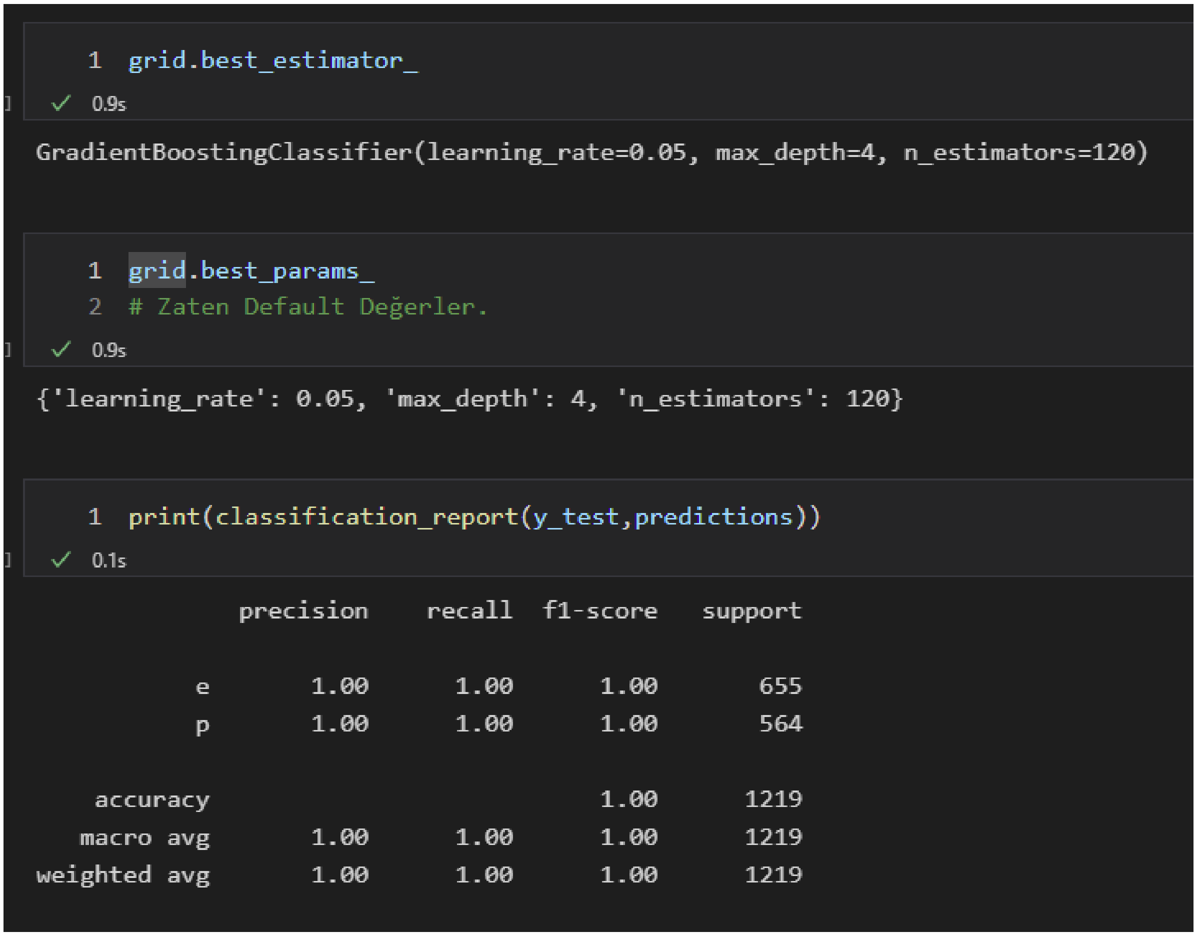
```python
predictions = grid.predict(X_test)
```

✓ 0.1s

```python
predictions
```

✓ 0.1s

```
array(['p', 'e', 'p', ..., 'p', 'p', 'e'], dtype=object)
```



```python
grid.best_estimator_
```

✓ 0.9s

```
GradientBoostingClassifier(learning_rate=0.05, max_depth=4, n_estimators=120)
```

```python
grid.best_params_
# Zaten Default Değerler.
```

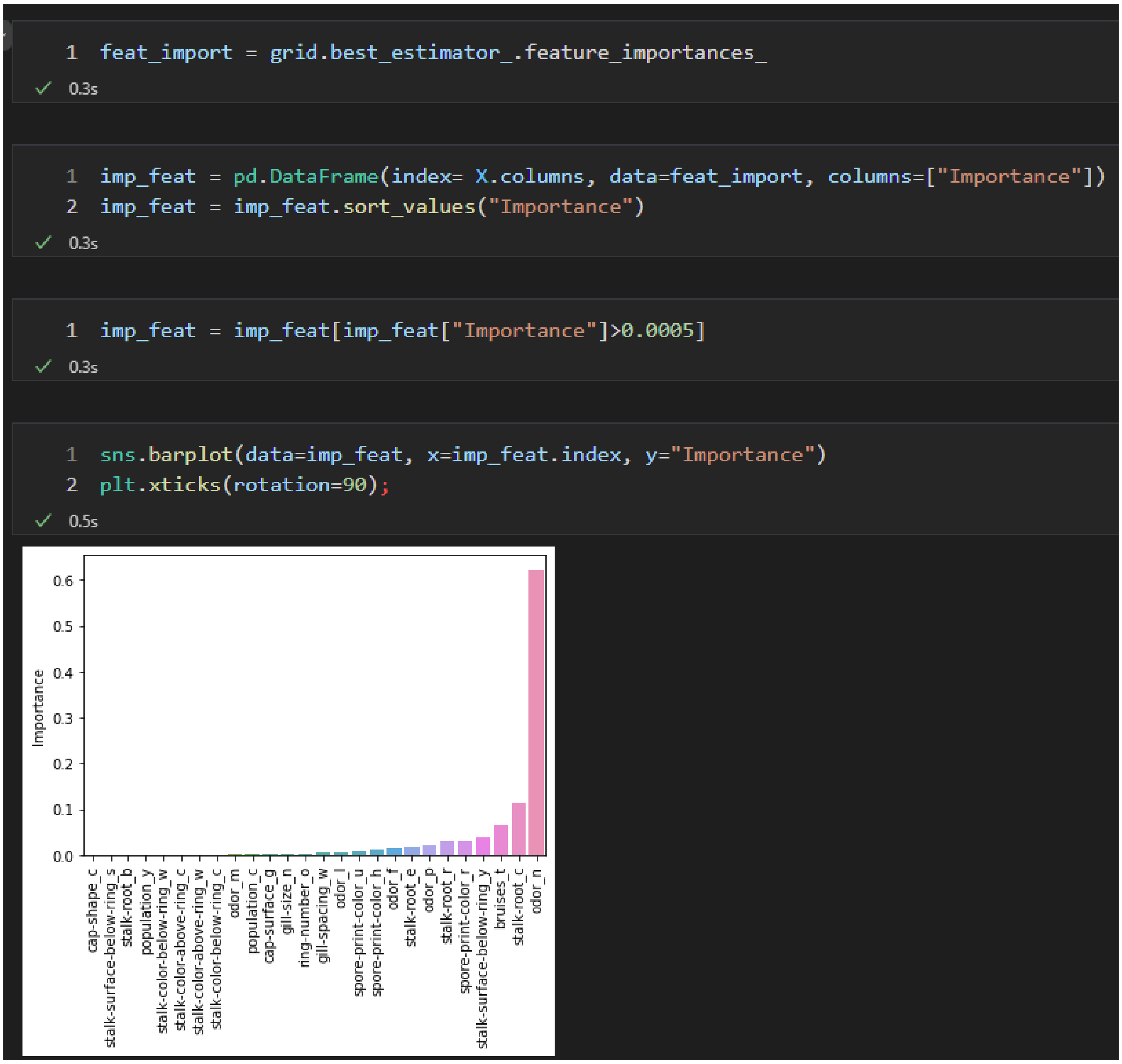
✓ 0.9s

```
{'learning_rate': 0.05, 'max_depth': 4, 'n_estimators': 120}
```

```python
print(classification_report(y_test,predictions))
```

✓ 0.1s

```
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       655
           p       1.00      1.00      1.00       564

    accuracy                           1.00      1219
   macro avg       1.00      1.00      1.00      1219
weighted avg       1.00      1.00      1.00      1219
```



```
feat_import = grid.best_estimator_.feature_importances_
```

✓ 0.3s

```
imp_feat = pd.DataFrame(index= X.columns, data=feat_import, columns=["Importance"])
imp_feat = imp_feat.sort_values("Importance")
```

✓ 0.3s

```
imp_feat = imp_feat[imp_feat["Importance"]>0.0005]
```

✓ 0.3s

```
sns.barplot(data=imp_feat, x=imp_feat.index, y="Importance")
plt.xticks(rotation=90);
```

✓ 0.5s