

Assignment 2, Data Mining

Murat Nargiza

Put all deliverables into github repository in your profile. Share link to google form 24 hours before defense. Defend by explaining deliverables and answering questions.

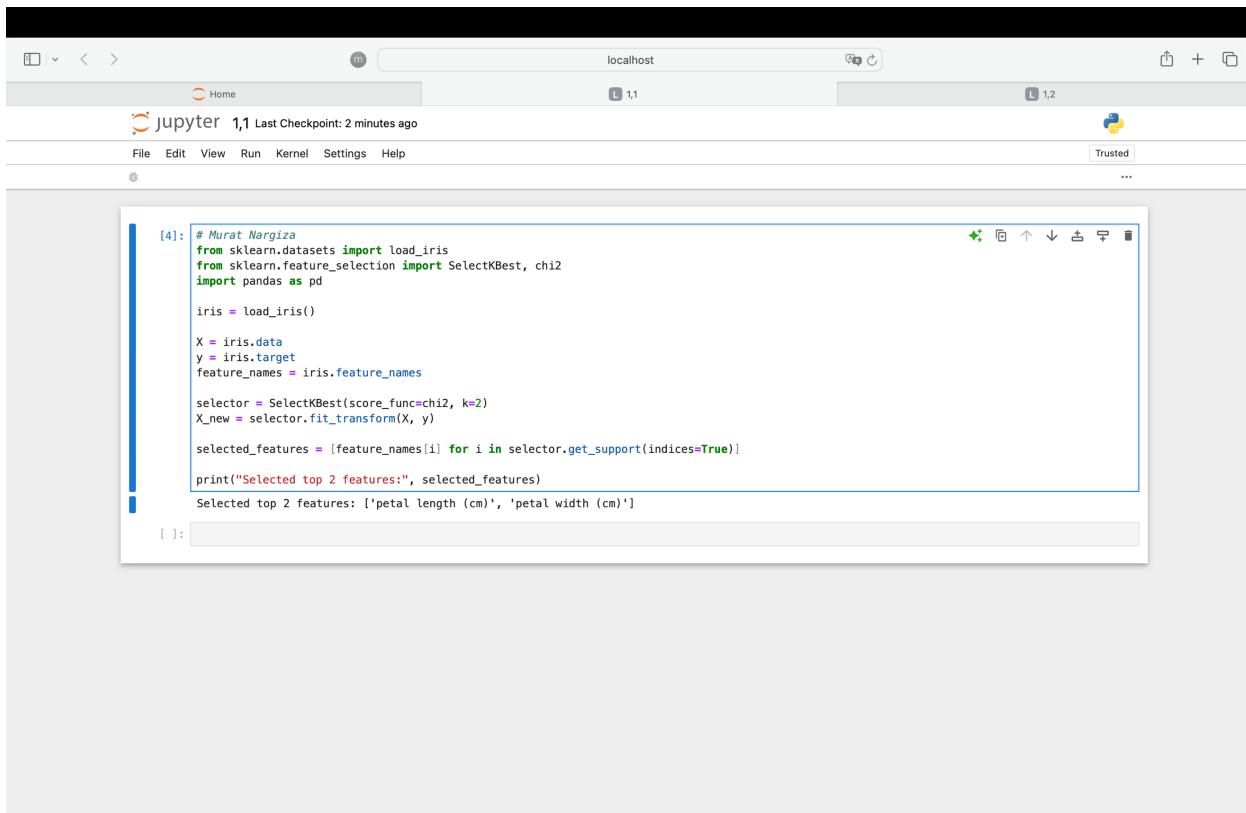
Deliverables: .ipynb

Google form: https://docs.google.com/forms/d/e/1FAIpQLSe0GyNdOYIvM1tX_I_CtIPod5jBf-ACLGdHYZq1gVZbUeBzlg/viewform?usp=sf_link

Exercise 1: Feature Selection with `SelectKBest`

Objective: Use `SelectKBest` from scikit-learn to select the top k features from a dataset.

1. Load the Iris dataset from scikit-learn.
2. Split the dataset into features and target variable.
3. Use `SelectKBest` with the `chi2` score function to select the top 2 features.
4. Print the selected feature names.



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Shows "localhost" and "1.2" in the top right.
- Toolbar:** Includes icons for file operations, a search bar, and a "Trusted" status.
- Cell 1 (In [1]):** Displays the URL "jupyter 1,1 Last Checkpoint: 2 minutes ago".
- Cell 2 (In [2]):** Contains the Python code for feature selection:

```
[4]: # Murat Nargiza
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, chi2
import pandas as pd

iris = load_iris()

X = iris.data
y = iris.target
feature_names = iris.feature_names

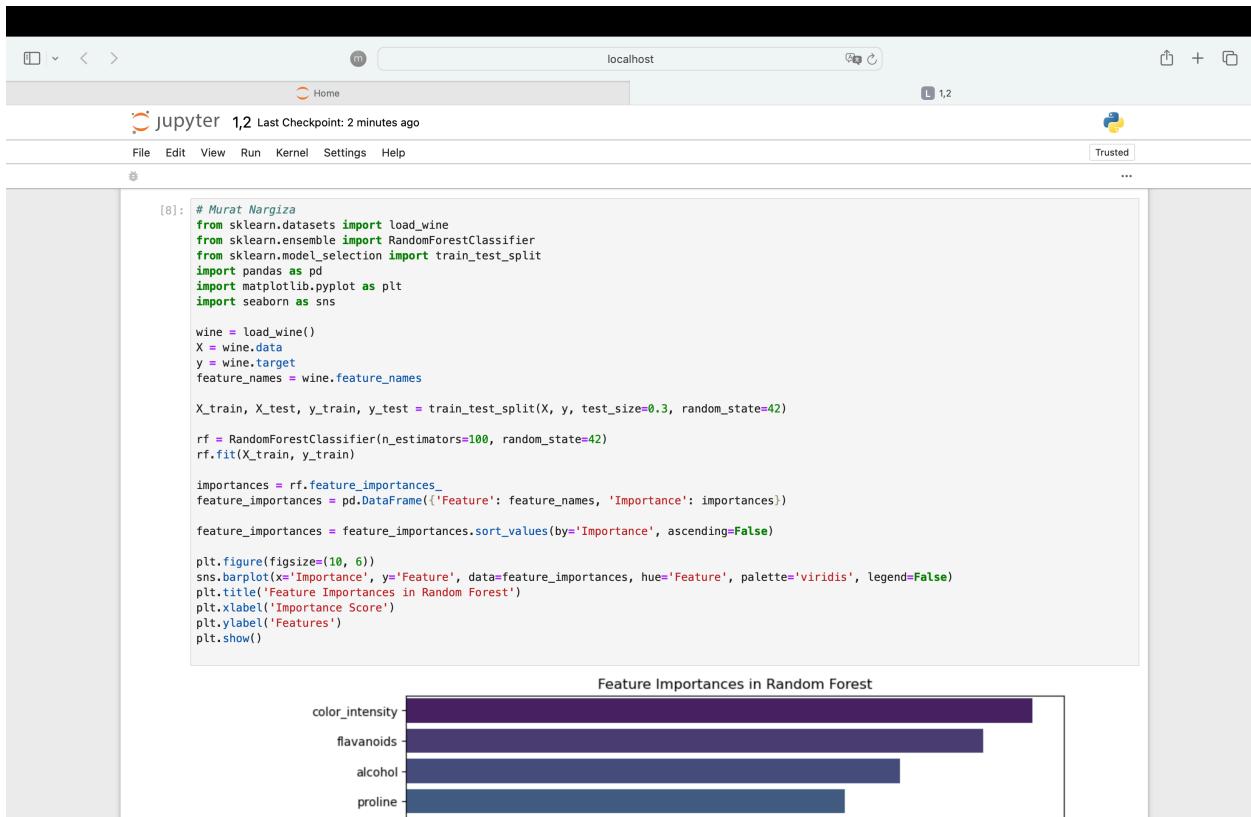
selector = SelectKBest(score_func=chi2, k=2)
X_new = selector.fit_transform(X, y)

selected_features = [feature_names[i] for i in selector.get_support(indices=True)]
print("Selected top 2 features:", selected_features)
```
- Cell 2 (Out [2]):** Shows the output of the code: "Selected top 2 features: ['petal length (cm)', 'petal width (cm)']".

Exercise 2: Feature Importance with Random Forest

Objective: Use a Random Forest classifier to determine feature importance.

1. Load the Wine dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train a Random Forest classifier on the training data.
4. Extract and visualize feature importances.



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code and a resulting visualization.

```
[8]: # Murat Nargiza
from sklearn.datasets import load_wine
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

wine = load_wine()
X = wine.data
y = wine.target
feature_names = wine.feature_names

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

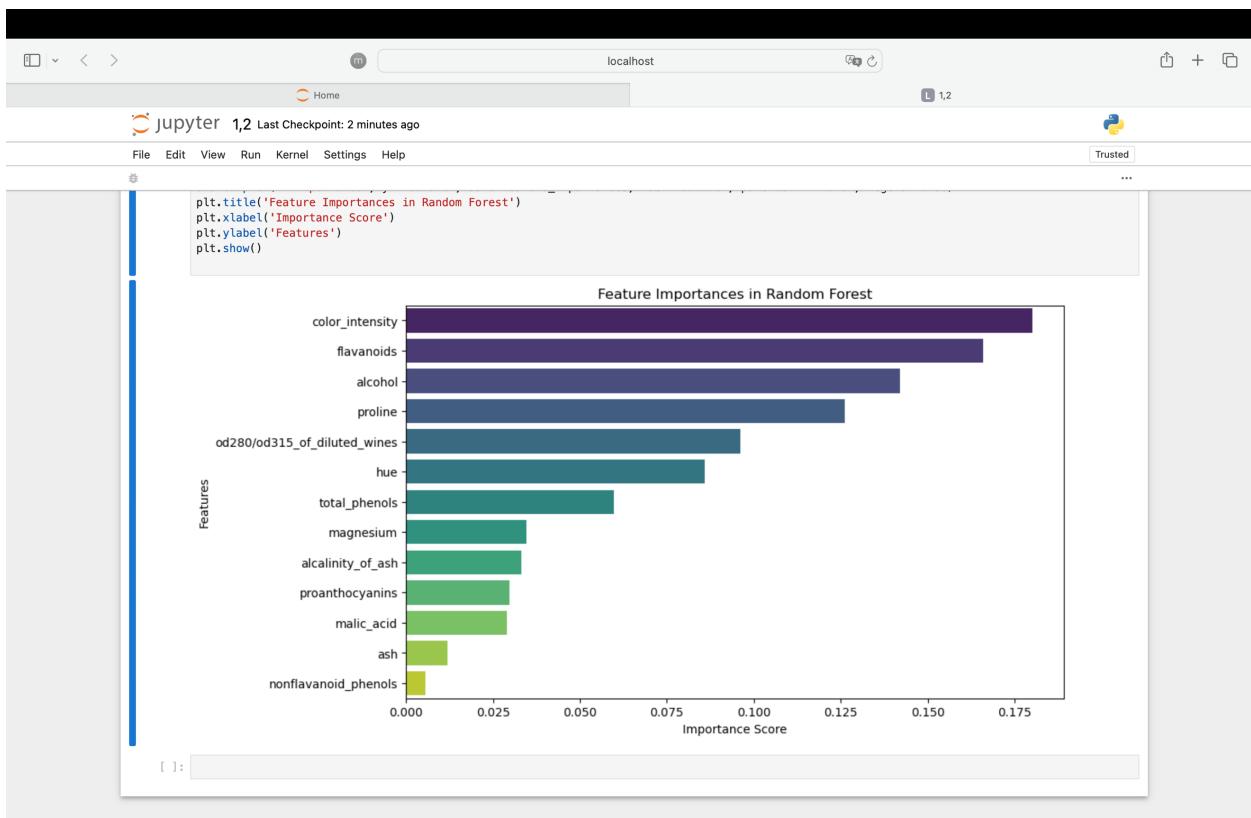
importances = rf.feature_importances_
feature_importances_ = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

feature_importances_ = feature_importances_.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importances_, hue='Feature', palette='viridis', legend=False)
plt.title('Feature Importances in Random Forest')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```

The visualization is a horizontal bar chart titled "Feature Importances in Random Forest". The x-axis is labeled "Importance Score" and ranges from 0 to approximately 0.25. The y-axis is labeled "Features" and lists four categories: "color_intensity", "flavonoids", "alcohol", and "proline". The bars are colored using a viridis color map, where darker shades represent higher importance. The "color_intensity" feature has the highest importance, followed by "flavonoids", "alcohol", and "proline".

Feature	Importance Score
color_intensity	~0.24
flavonoids	~0.18
alcohol	~0.15
proline	~0.12



Exercise 3: Recursive Feature Elimination (RFE)

Objective: Use Recursive Feature Elimination (RFE) to select features and evaluate model performance.

1. Load the Breast Cancer dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Use RFE with a Support Vector Machine (SVM) classifier to select features.
4. Train an SVM model with the selected features and evaluate its performance.

The screenshot shows a Jupyter Notebook interface running on localhost. The code cell [2] contains the following Python script:

```
# Murat Nargiza
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.feature_selection import RFE
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
X = cancer.data
y = cancer.target
feature_names = cancer.feature_names

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

svm = SVC(kernel="linear")
rfe = RFE(estimator=svm, n_features_to_select=10)
rfe.fit(X_train, y_train)

X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)

svm.fit(X_train_rfe, y_train)

y_pred = svm.predict(X_test_rfe)
accuracy = accuracy_score(y_test, y_pred)

selected_features = [feature_names[i] for i in rfe.get_support(indices=True)]
print("Selected features:", selected_features)
print("Model accuracy with selected features:", accuracy)

Selected features: ['mean radius', 'mean concavity', 'mean concave points', 'radius error', 'texture error', 'worst smoothness', 'worst compactness', 'worst concavity', 'worst concave points', 'worst symmetry']
Model accuracy with selected features: 0.9298245614035088
```

Exercise 4: L1 Regularization for Feature Selection

Objective: Use L1 regularization (Lasso) for feature selection.

1. Load the Diabetes dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Apply Lasso regression for feature selection.
4. Train a model using selected features and evaluate its performance.

The screenshot shows a Jupyter Notebook interface running on localhost, version 1.4. The code cell [2] contains Python code for performing Lasso regression on the diabetes dataset. The output shows the selected features and the mean squared error.

```
[2]: # Murat Nargiza
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd

diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target
feature_names = diabetes.feature_names

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)

coefficients = lasso.coef_
selected_features = np.array(feature_names)[coefficients != 0]

X_train_selected = X_train[:, coefficients != 0]
X_test_selected = X_test[:, coefficients != 0]

lasso_selected = Lasso(alpha=0.1)
lasso_selected.fit(X_train_selected, y_train)

y_pred = lasso_selected.predict(X_test_selected)
mse = mean_squared_error(y_test, y_pred)

print("Selected features:", selected_features)
print("Mean Squared Error with selected features:", mse)
Selected features: ['sex' 'bmi' 'bp' 's1' 's3' 's5' 's6']
Mean Squared Error with selected features: 2775.184056357289
```

Classification Exercises

Exercise 1: Logistic Regression

Objective: Build a logistic regression model to classify data.

1. Load the Iris dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train a logistic regression model on the training set.
4. Evaluate the model's performance on the test set using accuracy and a confusion matrix.

Jupyter 2.1 Last Checkpoint: 13 minutes ago

```
[2]: # Murat Nargiza
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

logreg = LogisticRegression(max_iter=200)
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

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

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d", cbar=False,
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

Accuracy: 1.00
Confusion Matrix:
[[19 0 0]
 [0 13 0]
 [0 0 13]]

Confusion Matrix

Jupyter 2.1 Last Checkpoint: 13 minutes ago

```
[2]: # Murat Nargiza
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

logreg = LogisticRegression(max_iter=200)
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

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

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d", cbar=False,
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

Accuracy: 1.00
Confusion Matrix:
[[19 0 0]
 [0 13 0]
 [0 0 13]]

Confusion Matrix

		setosa	versicolor	virginica
True Labels	setosa	19	0	0
	versicolor	0	13	0
virginica	0	0	13	

True Labels

Predicted Labels

Exercise 2: Support Vector Machine (SVM)

Objective: Use an SVM classifier to classify data.

1. Load the Breast Cancer dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train an SVM model on the training data.
4. Evaluate the model's performance on the test data using accuracy and a confusion matrix.

jupyter 2,2 Last Checkpoint: 3 seconds ago

```
[2]: # Murat Nargiza
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

breast_cancer = load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

y_pred = svm_model.predict(X_test)

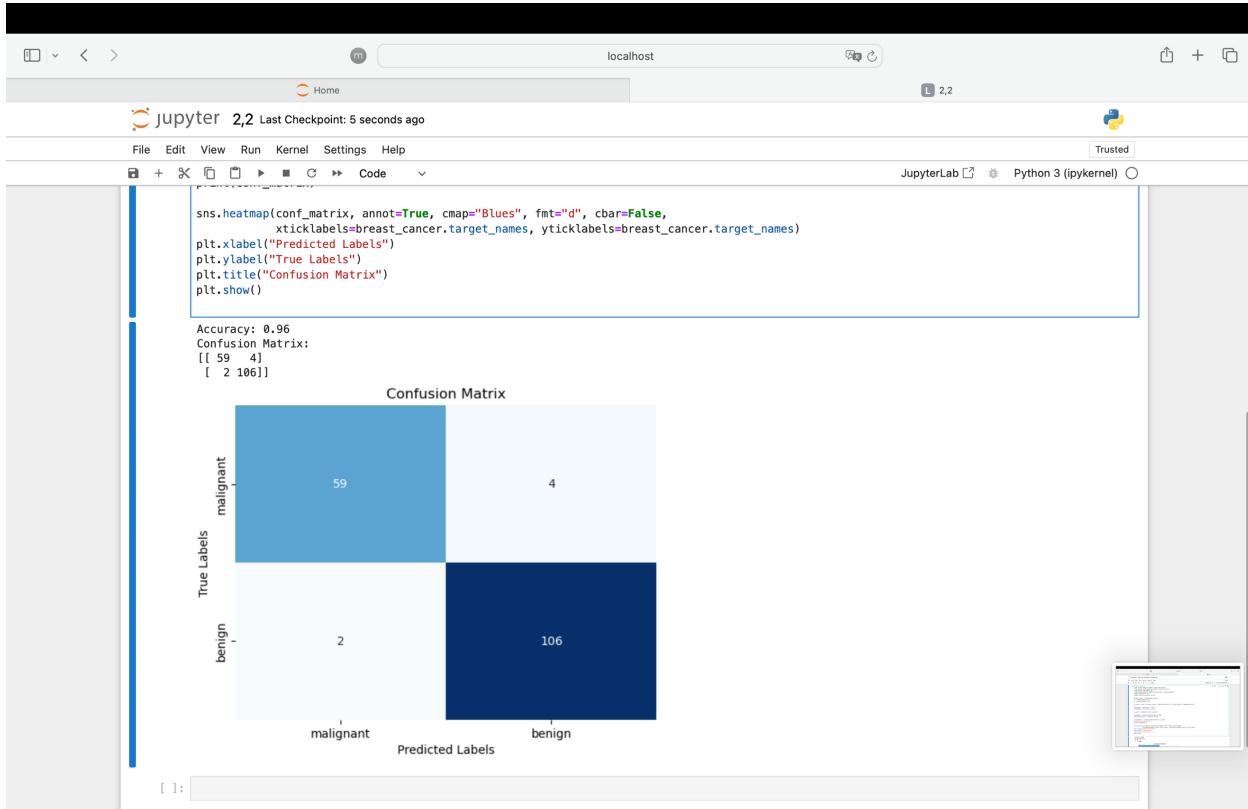
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d", cbar=False,
            xticklabels=breast_cancer.target_names, yticklabels=breast_cancer.target_names)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

Accuracy: 0.96
Confusion Matrix:
[[59 4]
 [2 106]]

Confusion Matrix



Exercise 3: Decision Tree Classifier

Objective: Build a decision tree classifier and visualize it.

1. Load the Wine dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train a decision tree classifier on the training set.
4. Visualize the decision tree.

Jupyter Notebook interface showing two code cells and their corresponding decision tree visualizations.

Code Cell 1:

```
[2]: # Murat Nargiza
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

wine = load_wine()
X = wine.data
y = wine.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)

plt.figure(figsize=(20,10))
plot_tree(tree_clf, filled=True, feature_names=wine.feature_names, class_names=wine.target_names, rounded=True)
plt.title("Decision Tree for Wine Dataset")
plt.show()
```

Decision Tree for Wine Dataset

```

graph TD
    Root["color_intensity <= 3.82  
gini = 0.658  
samples = 124  
value = [40, 50, 34]  
class = class_1"] -- True --> Node1["proline <= 1010.0  
gini = 0.083  
samples = 46  
value = [2, 44, 0]  
class = class_1"]
    Root -- False --> Node2["flavanoids <= 1.4  
gini = 0.567  
samples = 78  
value = [38, 6, 34]  
class = class_0"]
    Node1 -- True --> Node3["ash <= 3.07  
gini = 0.043  
samples = 45  
value = [1, 44, 0]  
class = class_1"]
    Node1 -- False --> Node4["gini = 0.0  
samples = 1  
value = [1, 0, 0]  
class = class_0"]
    Node2 -- True --> Node5["gini = 0.0  
samples = 34  
value = [0, 0, 34]  
class = class_2"]
    Node2 -- False --> Node6["proline <= 724.5  
gini = 0.236  
samples = 44  
value = [38, 6, 0]  
class = class_0"]
    Node3 -- True --> Node7["gini = 0.0  
samples = 44  
value = [0, 44, 0]  
class = class_1"]
    Node3 -- False --> Node8["gini = 0.0  
samples = 1  
value = [1, 0, 0]  
class = class_0"]
    Node4 -- True --> Node9["alcohol <= 13.145  
gini = 0.245  
samples = 7  
value = [1, 6, 0]  
class = class_1"]
    Node4 -- False --> Node10["gini = 0.0  
samples = 6  
value = [0, 6, 0]  
class = class_1"]
    Node5 -- True --> Node11["gini = 0.0  
samples = 37  
value = [37, 0, 0]  
class = class_0"]
    Node5 -- False --> Node12["gini = 0.0  
samples = 1  
value = [1, 0, 0]  
class = class_0"]

```

Code Cell 2:

```
plt.show()
```

Decision Tree for Wine Dataset

```

graph TD
    Root["color_intensity <= 3.82  
gini = 0.658  
samples = 124  
value = [40, 50, 34]  
class = class_1"] -- True --> Node1["proline <= 1010.0  
gini = 0.083  
samples = 46  
value = [2, 44, 0]  
class = class_1"]
    Root -- False --> Node2["flavanoids <= 1.4  
gini = 0.567  
samples = 78  
value = [38, 6, 34]  
class = class_0"]
    Node1 -- True --> Node3["ash <= 3.07  
gini = 0.043  
samples = 45  
value = [1, 44, 0]  
class = class_1"]
    Node1 -- False --> Node4["gini = 0.0  
samples = 1  
value = [1, 0, 0]  
class = class_0"]
    Node2 -- True --> Node5["gini = 0.0  
samples = 34  
value = [0, 0, 34]  
class = class_2"]
    Node2 -- False --> Node6["proline <= 724.5  
gini = 0.236  
samples = 44  
value = [38, 6, 0]  
class = class_0"]
    Node3 -- True --> Node7["gini = 0.0  
samples = 44  
value = [0, 44, 0]  
class = class_1"]
    Node3 -- False --> Node8["gini = 0.0  
samples = 1  
value = [1, 0, 0]  
class = class_0"]
    Node4 -- True --> Node9["alcohol <= 13.145  
gini = 0.245  
samples = 7  
value = [1, 6, 0]  
class = class_1"]
    Node4 -- False --> Node10["gini = 0.0  
samples = 6  
value = [0, 6, 0]  
class = class_1"]
    Node5 -- True --> Node11["gini = 0.0  
samples = 37  
value = [37, 0, 0]  
class = class_0"]
    Node5 -- False --> Node12["gini = 0.0  
samples = 1  
value = [1, 0, 0]  
class = class_0"]

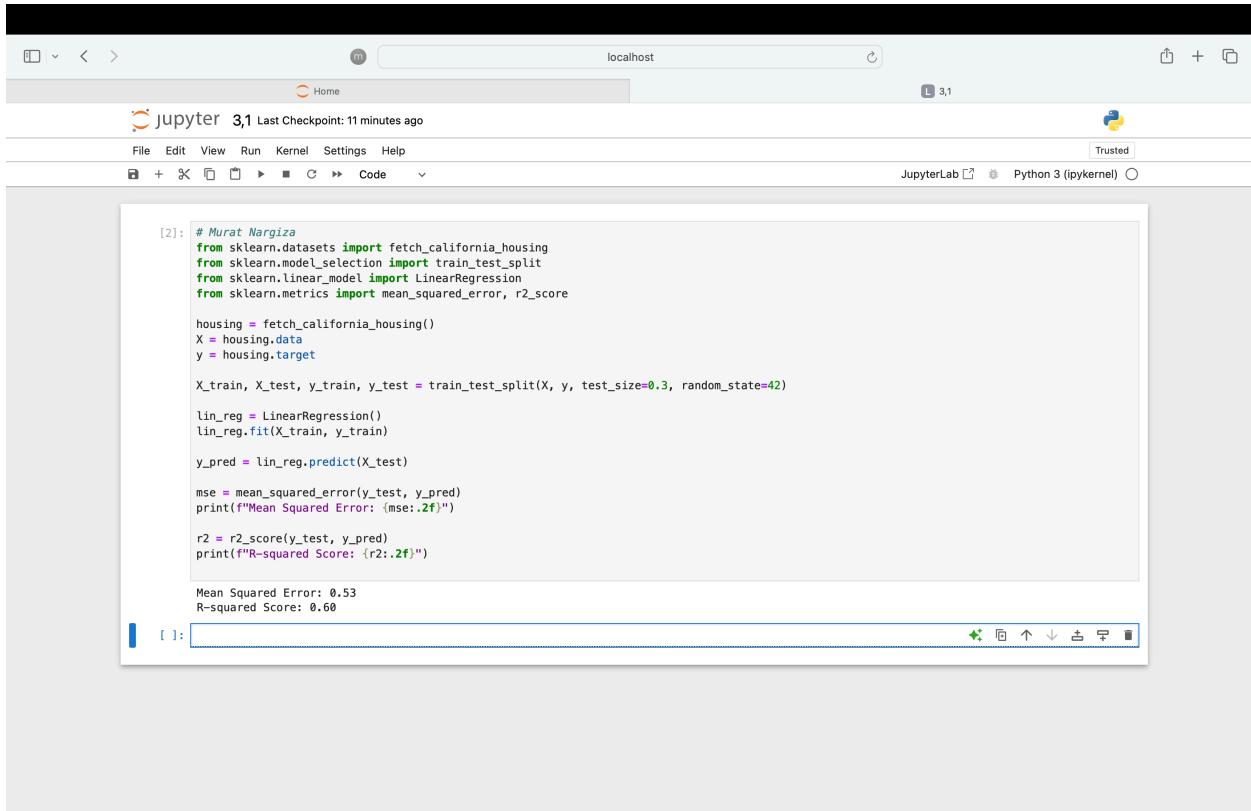
```

Regression Exercises

Exercise 1: Linear Regression

Objective: Build a linear regression model to predict a continuous target variable.

1. Load the Boston Housing dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train a linear regression model on the training set.
4. Evaluate the model's performance using mean squared error (MSE) and R-squared score.



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter 3.1 Last Checkpoint: 11 minutes ago
- Toolbar:** File Edit View Run Kernel Settings Help
- Cell Status:** Trusted
- Code Cell Content:**

```
[2]: # Murat Nargiza
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

housing = fetch_california_housing()
X = housing.data
y = housing.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

y_pred = lin_reg.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")

r2 = r2_score(y_test, y_pred)
print(f"R-squared Score: {r2:.2f}")

Mean Squared Error: 0.53
R-squared Score: 0.60
```
- Cell Number:** [2]:
- Bottom Navigation:** A toolbar with various icons for navigating between cells and notebooks.

Exercise 2: Ridge Regression

Objective: Use Ridge regression to perform regularized linear regression.

1. Load the Diabetes dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train a Ridge regression model on the training set.
4. Evaluate the model's performance using mean squared error (MSE) and R-squared score.

The screenshot shows a Jupyter Notebook interface running on localhost. The notebook has a single cell containing Python code for performing Ridge regression on the diabetes dataset. The code imports necessary libraries, loads the dataset, splits it into training and testing sets, trains a Ridge regressor, and prints the Mean Squared Error and R-squared score. The output of the cell shows the calculated values.

```
[2]: # Murat Nargiza
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(X_train, y_train)

y_pred = ridge_reg.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")

r2 = r2_score(y_test, y_pred)
print(f"R-squared Score: {r2:.2f}")

Mean Squared Error: 3112.97
R-squared Score: 0.42
```

Exercise 3: Decision Tree Regression

Objective: Build a decision tree regression model and visualize it.

1. Load the Boston Housing dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train a decision tree regressor on the training set.
4. Evaluate the model's performance using mean squared error (MSE).
5. Visualize the decision tree.

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** "jupyter 3,3 Last Checkpoint: 5 seconds ago".
- Toolbar:** File, Edit, View, Run, Kernel, Settings, Help.
- Header:** Trusted, JupyterLab, Python 3 (ipykernel).
- Code Cell:** Contains Python code for a Decision Tree Regression model using scikit-learn. The code imports necessary libraries, fetches California housing data, splits it into training and testing sets, trains a DecisionTreeRegressor, calculates Mean Squared Error, and plots the decision tree.

```
[*:]: # Murat Nargiza
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn import tree

housing = fetch_california_housing()
X = housing.data
y = housing.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')

plt.figure(figsize=(20, 10))
tree.plot_tree(regressor, filled=True, feature_names=housing.feature_names, rounded=True)
plt.title('Decision Tree Regression Model')
plt.show()
```

- Output Cell:** Shows the status [:]: | and a set of navigation icons.