Report of assignment 2

1. **Feature Selection with `SelectKBest`**

```python
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, chi2
import pandas as pd

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

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

selected_features = selector.get_support(indices=True)
feature_names = [iris.feature_names[i] for i in selected_features]

print("Selected features:", feature_names)
```

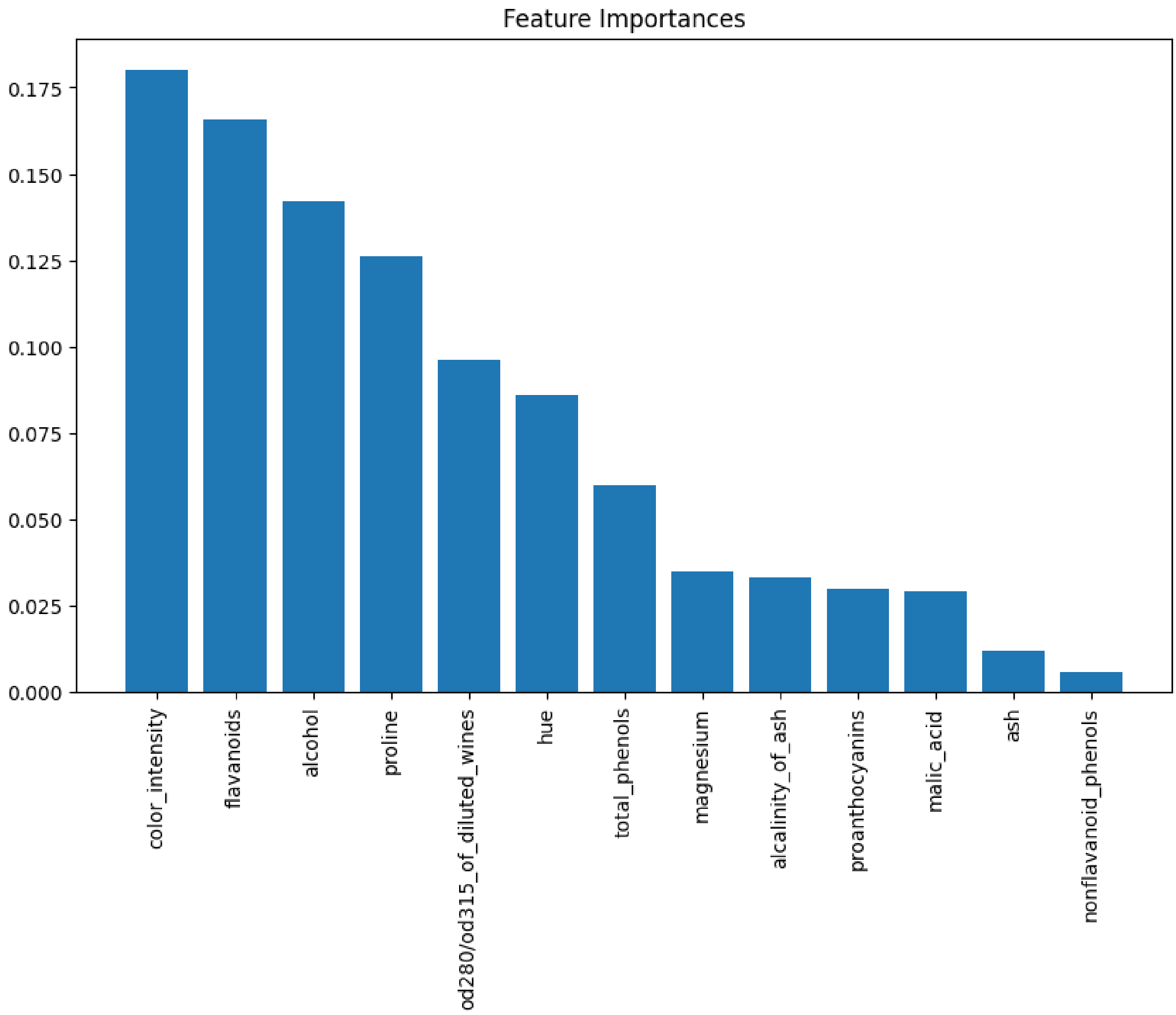
```
Selected features: ['petal length (cm)', 'petal width (cm)']
```

Steps:

1. Load Iris data with **load_iris()**
2. **SelectKBest** and **chi2** are used for feature selection based on the Chi-squared statistical test.
3. Fitting the selector and transforming the data
4. Getting the selected features' indices
5. Getting the feature names
6. Output

2. Feature Importance with Random Forest

```python
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

wine = load_wine()
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3, random_state=42)

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

importances = rf.feature_importances_
indices = importances.argsort()[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.bar(range(X_train.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), [wine.feature_names[i] for i in indices], rotation=90)
plt.show()
```



Steps:

1. Loading the Wine dataset using **load_wine()**
2. Splitting the dataset
3. Calculating feature importance
4. Plotting the feature importances

3. Recursive Feature Elimination (RFE)

```python
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3, random_state=42)

svc = SVC(kernel="linear", random_state=42)
rfe = RFE(estimator=svc, n_features_to_select=10)
rfe.fit(X_train, y_train)

y_pred = rfe.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy with RFE-selected features:", accuracy)
```

```
Accuracy with RFE-selected features: 0.9298245614035088
```

Steps:

1. Loading the Dataset with **load_breast_cancer()**
2. Splitting the Dataset
3. Creating an SVM Classifier
4. Feature Selection Using RFE
5. Prediction and Accuracy Evaluation

4. L1 Regularization for Feature Selection

```python
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


diabetes = load_diabetes()
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.target, test_size=0.3, random_state=42)

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

y_pred = lasso.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error with Lasso:", mse)
```

```
Mean Squared Error with Lasso: 2775.165076183445
```

Steps:

1. Loading the Iris Dataset
2. Splitting the Dataset
3. Creating and Training the Lasso Model
4. Making Predictions
5. Evaluating the Model

Classification Exercises

1. **Logistic Regression**

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3, random_state=42)

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

y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 1.0
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

Steps:

1. Loading the Iris Dataset
2. Splitting the Dataset
3. Creating and Training the Logistic Regression Model
4. Making Predictions
5. Evaluating the Model

2. **Support Vector Machine (SVM)**

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3, random_state=42)

svm = SVC(kernel="linear")
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

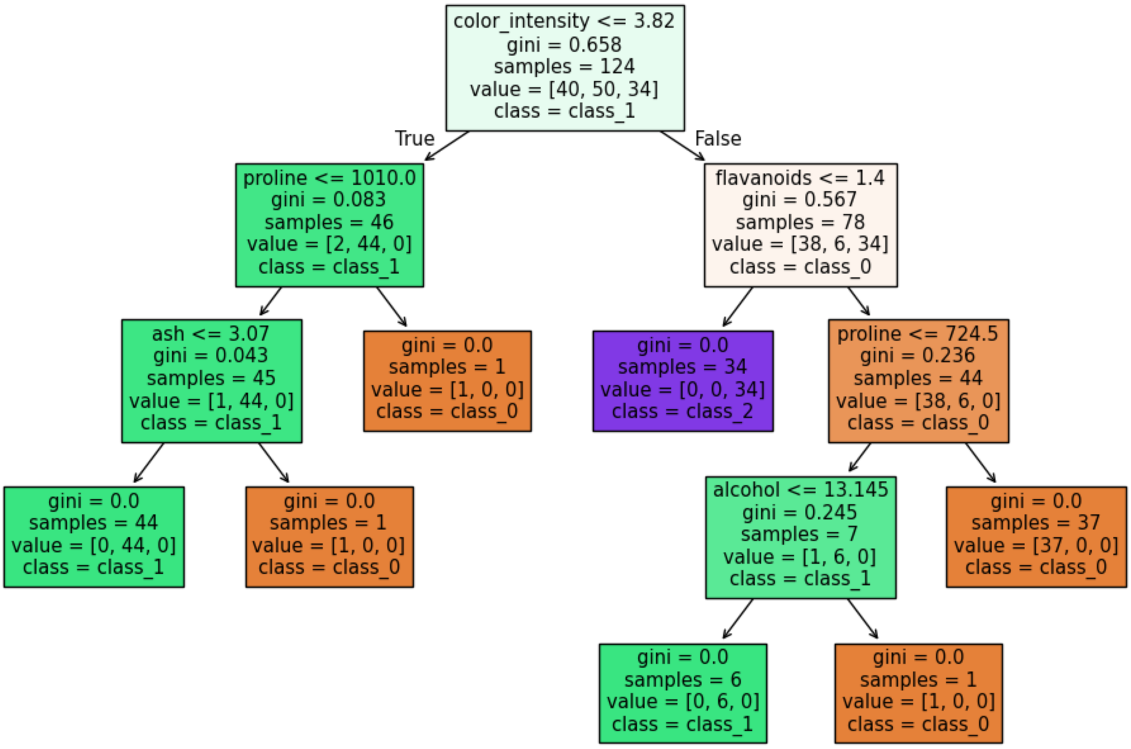
```
Accuracy: 0.9649122807017544
Confusion Matrix:
 [[ 59   4]
 [  2 106]]
```

Steps:

1. Loading the Breast Cancer Dataset
2. Splitting the Dataset
3. Creating and Training the SVM Model
4. Making Predictions
5. Evaluating the Model

3. Decision Tree Classifier

```python
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

wine = load_wine()
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3, random_state=42)

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

plt.figure(figsize=(12, 8))
plot_tree(tree, feature_names=wine.feature_names, class_names=wine.target_names, filled=True)
plt.show()
```



Steps:

1. Loading the Wine Dataset
2. Splitting the Dataset
3. Creating and Training the Decision Tree Model
4. Visualizing the Decision Tree

Regression Exercises

1: Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

housing = (variable) X_test: Any g()
X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.3, random_state=42)

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

y_pred = lr.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared:", r2_score(y_test, y_pred))
```

```
Mean Squared Error: 0.5305677824766757
R-squared: 0.595770232606166
```

Steps:

1. Load the Boston Housing dataset
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.

2: Ridge Regression

```
from sklearn.linear_model import Ridge
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

housing = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.3, random_state=42)

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

y_pred_ridge = ridge.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_ridge))
print("R-squared:", r2_score(y_test, y_pred_ridge))
```

```
Mean Squared Error: 0.5305052690933701
R-squared: 0.5958178603951634
```

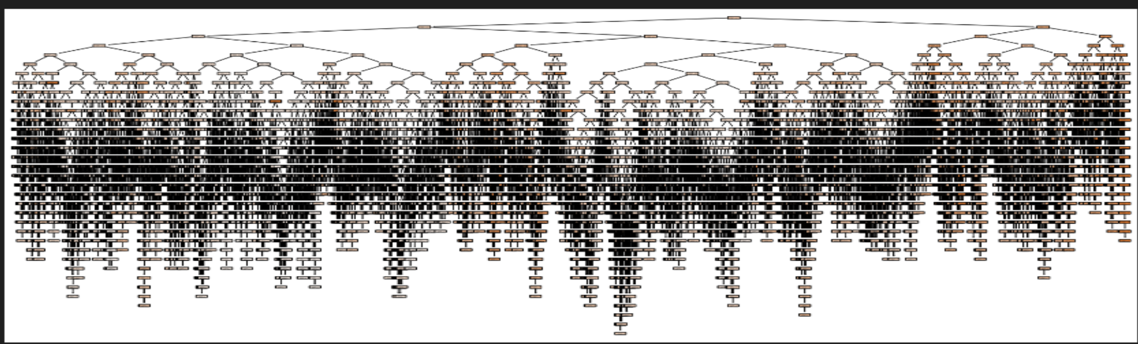
Steps:

1. Load the Diabetes dataset
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.

3: Decision Tree Regression

```python
# Import necessary libraries
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
from sklearn import tree
import matplotlib.pyplot as plt

# Load the California Housing dataset
housing = fetch_california_housing()
X = housing.data
y = housing.target

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

# Create a Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)

# Train the model
regressor.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = regressor.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Feature names for the California housing dataset
feature_names = housing.feature_names  # Manually specify the feature names

# Visualize the decision tree
plt.figure(figsize=(20,10))
tree.plot_tree(regressor, feature_names=feature_names, filled=True, rounded=True)
plt.show()
```

```
Mean Squared Error: 0.495235205629094
```



Steps:

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.