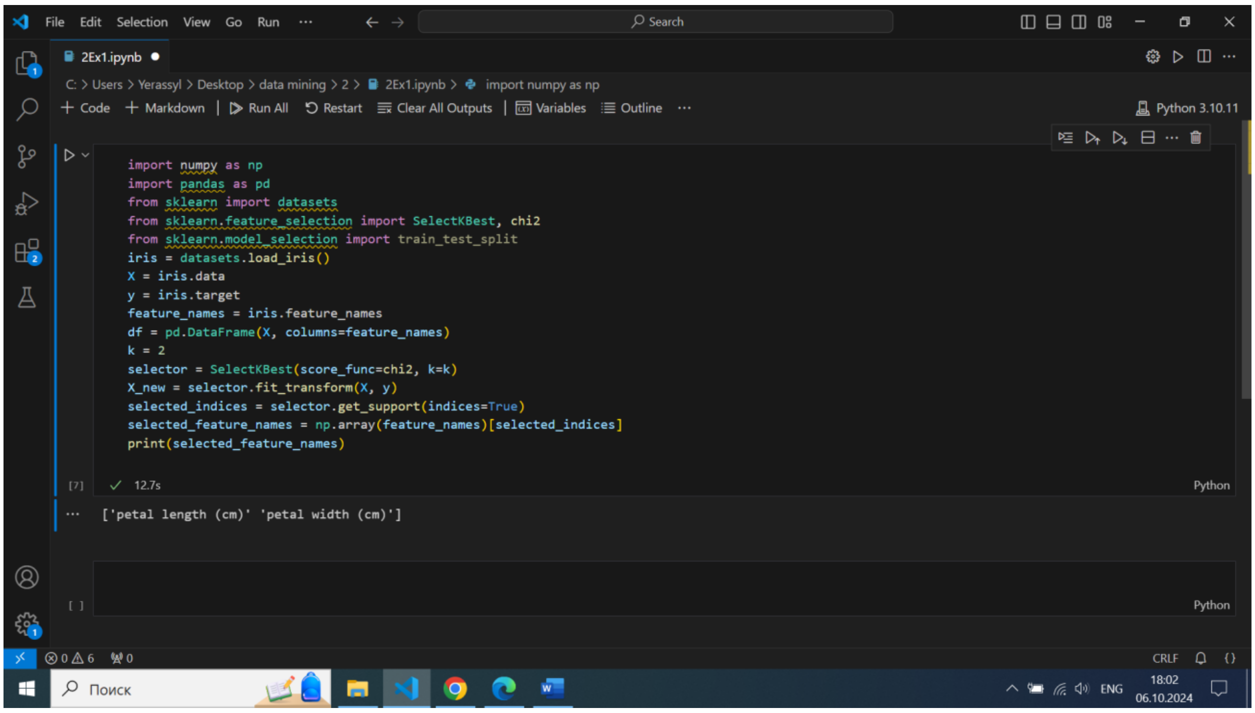
Exercise 1: Feature Selection with `SelectKBest`

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.

In the first exercise we needed to use SelectKBest in order to highlight the features from the dataset.



```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import train_test_split
iris = datasets.load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
df = pd.DataFrame(X, columns=feature_names)
k = 2
selector = SelectKBest(score_func=chi2, k=k)
X_new = selector.fit_transform(X, y)
selected_indices = selector.get_support(indices=True)
selected_feature_names = np.array(feature_names)[selected_indices]
print(selected_feature_names)
```

```
['petal length (cm)' 'petal width (cm)']
```

Steps:

1) We used numpy and pandas for working with data. Also, we used sklearn library for selection of features from data and for splitting the dataset for features and targets.
2) The Iris dataset we loaded form **datasets.load_iris()** method and it contains the features as petal length, width, sepal length, width.
3) Next, we splitted the data on x and y as the features and target accordingly.
4) In the task it was necessary to find 2 features and we set it using k=2.
5) **selector = SelectKBest(score_func=chi2, k=k)** shows that we used chi-squared score for featuring k aspects of dataset. **selector.fit_transform(X, y):** This fits the selector on the feature data (X) and transforms it to keep only the top k features.
6) And there we print the selected feature names that stores the features. Results are petal length and width.

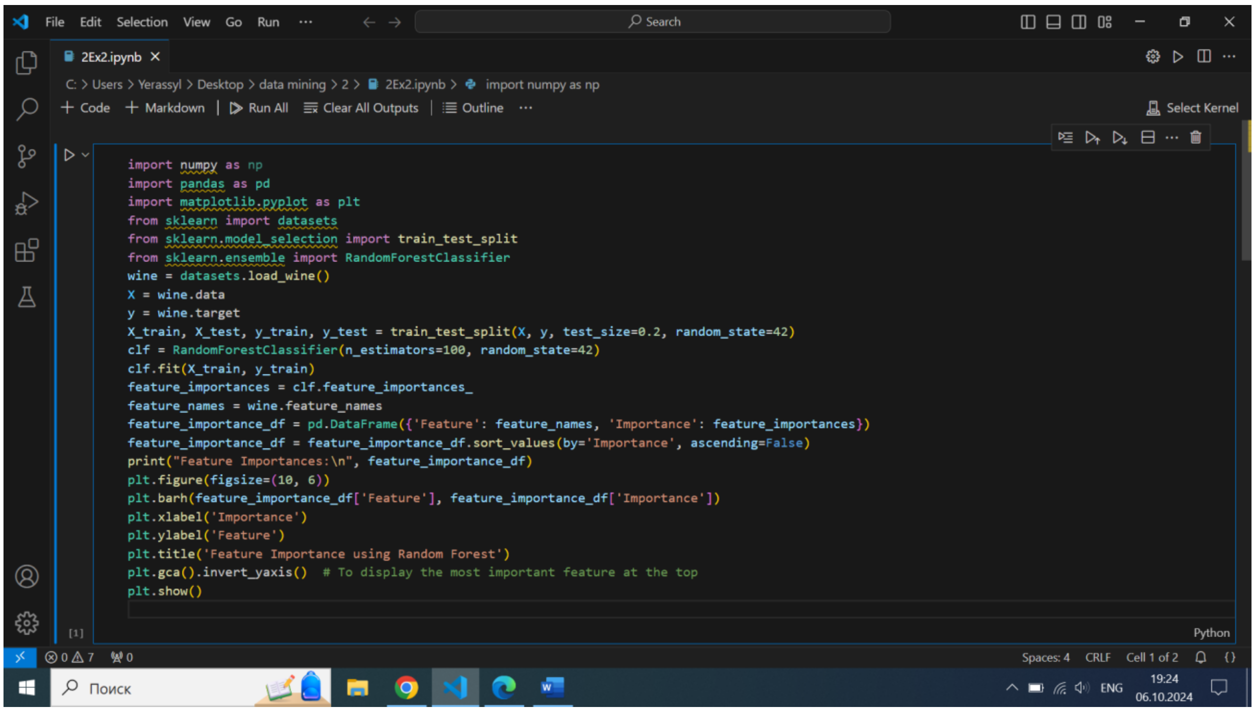
Exercise 2: Feature Importance with Random Forest

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

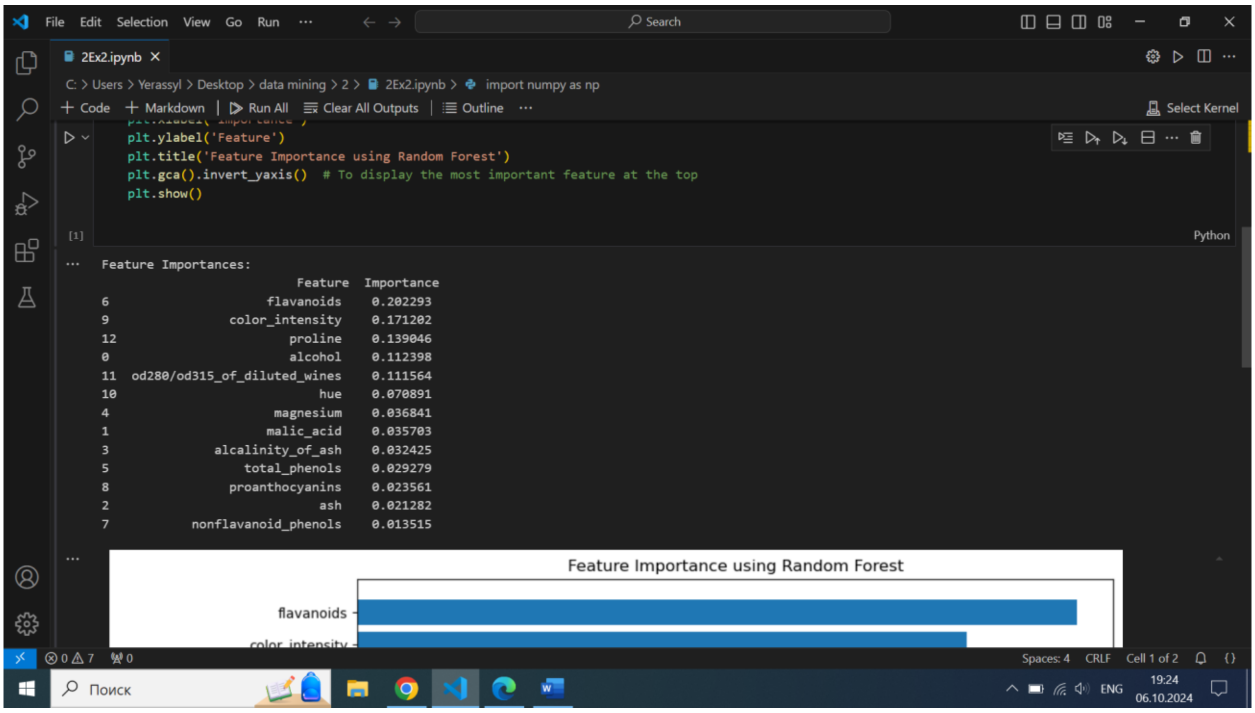
1. Load the Wine dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train a Random Forest classifier on the training data.

4. Extract and visualize feature importances.

In this task we need to use Random forest to measure the feature importances in Wine dataset.

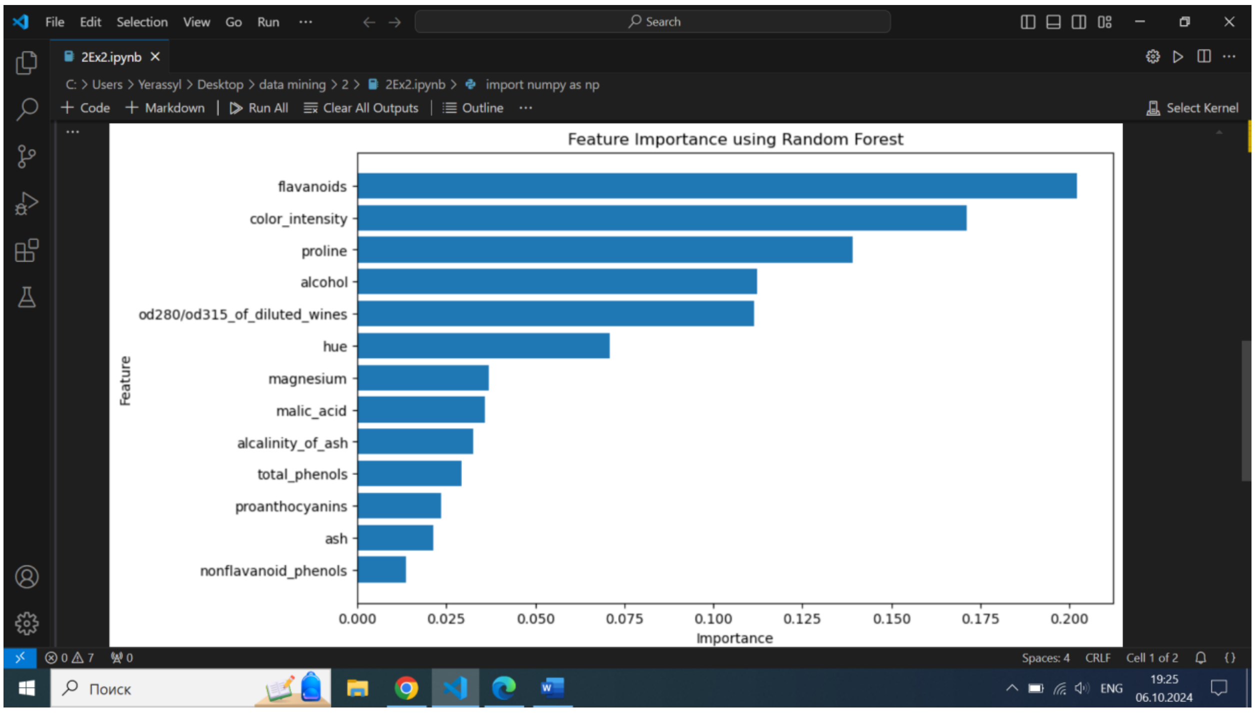

File Edit Selection View Go Run
2Ex2.ipynb
C: > Users > Yerassyl > Desktop > data mining > 2 > 2Ex2.ipynb > import numpy as np
+ Code + Markdown | Run All Clear All Outputs | Outline
Select Kernel
Python
Поиск

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
wine = datasets.load_wine()
X = wine.data
y = wine.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
feature_importances = clf.feature_importances_
feature_names = wine.feature_names
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
print("Feature Importances:\n", feature_importance_df)
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance using Random Forest')
plt.gca().invert_yaxis()  # To display the most important feature at the top
plt.show()
```


File Edit Selection View Go Run
2Ex2.ipynb
C: > Users > Yerassyl > Desktop > data mining > 2 > 2Ex2.ipynb > import numpy as np
+ Code + Markdown | Run All Clear All Outputs | Outline
Select Kernel
Python
Feature Importance using Random Forest
flavanoids
color_intensity
Поиск

```python
plt.ylabel('Feature')
plt.title('Feature Importance using Random Forest')
plt.gca().invert_yaxis()  # To display the most important feature at the top
plt.show()
```

```
Feature Importances:
                          Feature  Importance
6                     flavanoids    0.202293
9                color_intensity    0.171202
12                       proline    0.139046
0                        alcohol    0.112398
11  od280/od315_of_diluted_wines    0.111564
10                           hue    0.070891
4                      magnesium    0.036841
1                     malic_acid    0.035703
3              alcalinity_of_ash    0.032425
5                  total_phenols    0.029279
8                proanthocyanins    0.023561
2                            ash    0.021282
7           nonflavanoid_phenols    0.013515
```



Steps:

1) We use library **numpy, pandas, sklearn** libraries as previous task. Alse, there we should use **matplotlib.pyplot** library for the visualization and **RandomForestClassifier** as the machine learning model classifying tasks.
2) **X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)** uses for training and testing the datas**.**
3) **clf = RandomForestClassifier(n_estimators=100, random_state=42**) initialize the random forest classifier as number of trees as 100 and random state 42. **clf.fit(X_train, y_train)** trains the model by training data**.**
4) Next, we create dataframe with 2 columns as feature and importance. **sort_values(by='Importance', ascending=False)** sorts the dataframe importance aspect by descending order.
5) We set the bar chart by the plt methods that helps manipulate with graph data.
6) In the result, we see the graph.

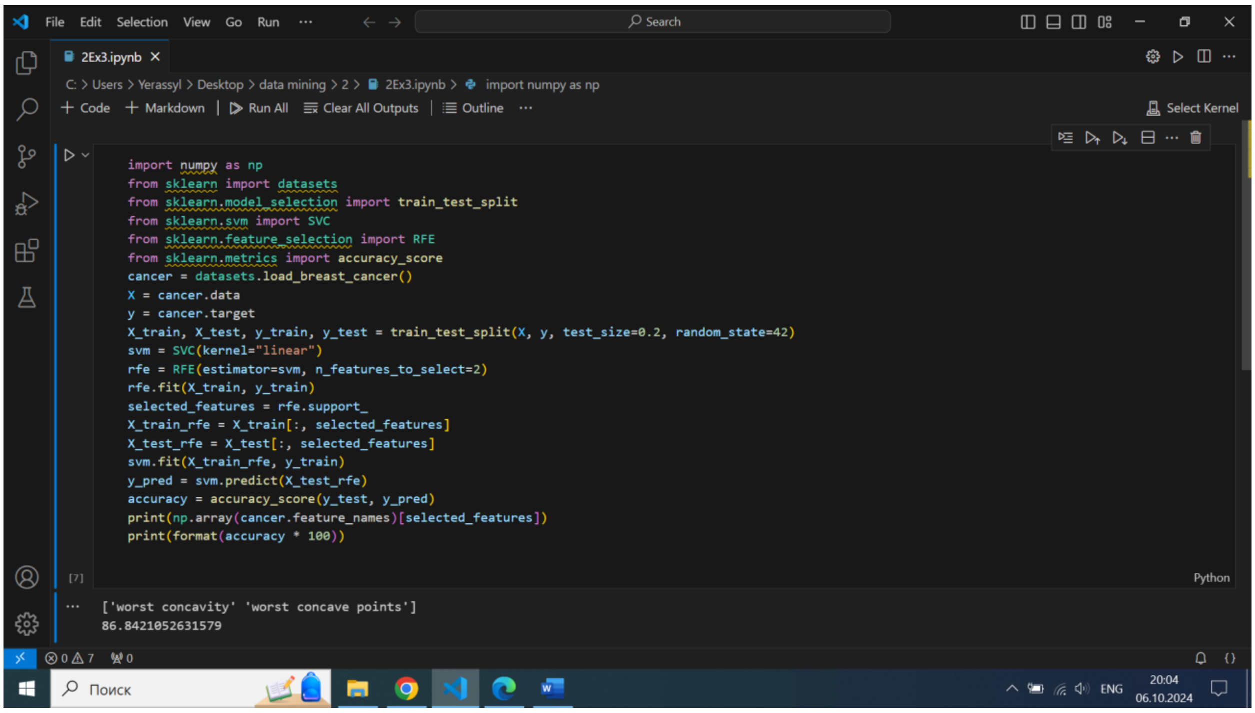
Exercise 3: Recursive Feature Elimination (RFE)

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.

In this task we use another feature selection method RFE


File Edit Selection View Go Run
2Ex3.ipynb
C: > Users > Yerassyl > Desktop > data mining > 2 > 2Ex3.ipynb > import numpy as np
+ Code + Markdown | Run All Clear All Outputs | Outline
Select Kernel
[7]
Python

```
import numpy as np
from sklearn import datasets
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 = datasets.load_breast_cancer()
X = cancer.data
y = cancer.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
svm = SVC(kernel="linear")
rfe = RFE(estimator=svm, n_features_to_select=2)
rfe.fit(X_train, y_train)
selected_features = rfe.support_
X_train_rfe = X_train[:, selected_features]
X_test_rfe = X_test[:, selected_features]
svm.fit(X_train_rfe, y_train)
y_pred = svm.predict(X_test_rfe)
accuracy = accuracy_score(y_test, y_pred)
print(np.array(cancer.feature_names)[selected_features])
print(format(accuracy * 100))
```

```
['worst concavity' 'worst concave points']
86.8421052631579
```

Steps:

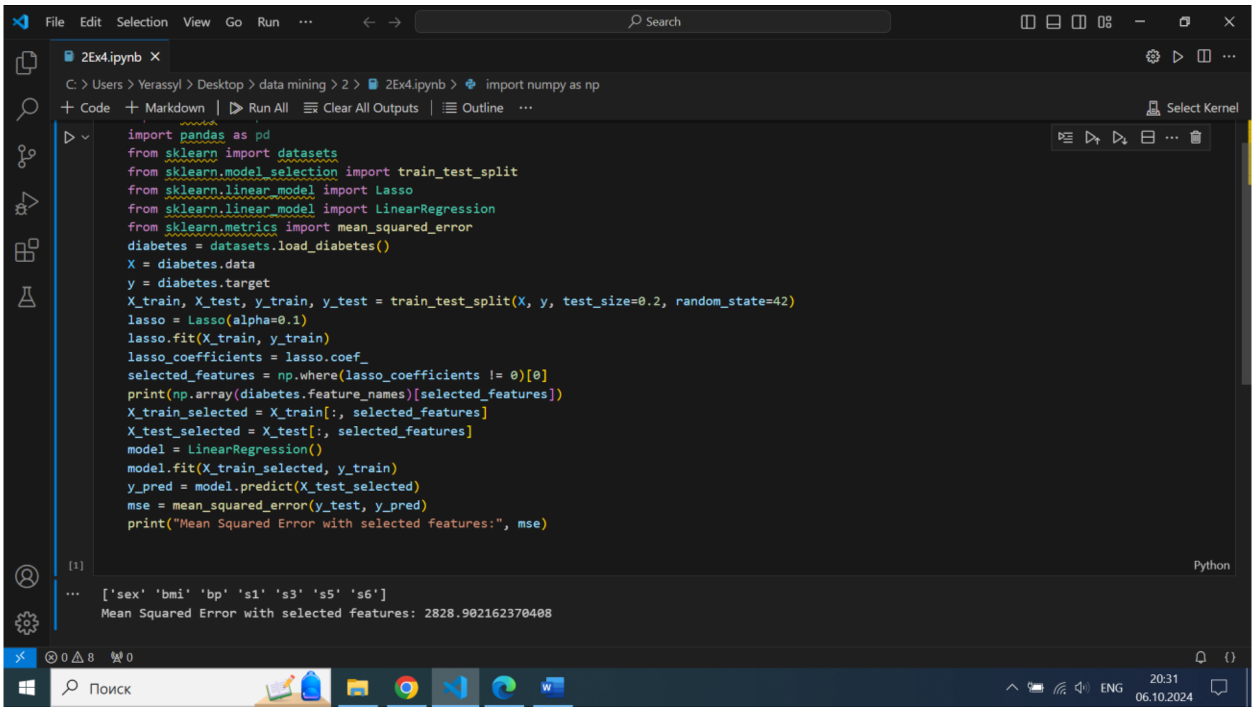
1) As the previous task we use libraries and there we have additionally RFE and SVC.
2) We load the dataset of cancer and get the x and y feature and target.
3) Split them as previously.
4) We **rfe = RFE(estimator=svm, n_features_to_select=2)** choose the 2 features using svm.
5) **rfe.fit(X_train, y_train)** using rfe model to determination of importance of the features.
6) **selected_features** is a boolean array indicating which features were selected (True) and which were not (False).
7) **svm.fit(X_train_rfe, y_train)** the SVM model is trained on the reduced training dataset.
8) **y_pred = svm.predict(X_test_rfe**) make predictions and get the result in percentage.

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.

Using the another feature selection method as L1 regularization(Lasso).



```python
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
lasso_coefficients = lasso.coef_
selected_features = np.where(lasso_coefficients != 0)[0]
print(np.array(diabetes.feature_names)[selected_features])
X_train_selected = X_train[:, selected_features]
X_test_selected = X_test[:, selected_features]
model = LinearRegression()
model.fit(X_train_selected, y_train)
y_pred = model.predict(X_test_selected)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error with selected features:", mse)
```

```
['sex' 'bmi' 'bp' 's1' 's3' 's5' 's6']
Mean Squared Error with selected features: 2828.902162370408
```

Steps:

1) Using new libraries Lasso, LinearRegression and mean_squared_error.
2) Also, load the dataset and split it as previous tasks.
3) **Lasso.fit(x_train,y_train)** trains the datas.
4) **mse = mean_squared_error(y_test, y_pred)** show that the mean squared error (MSE) is calculated by comparing the predicted values (y_pred) with the true values (y_test). MSE measures the average squared difference between the predicted and actual values, with lower values indicating better model performance.
5) Printing the results.

Classification Exercises

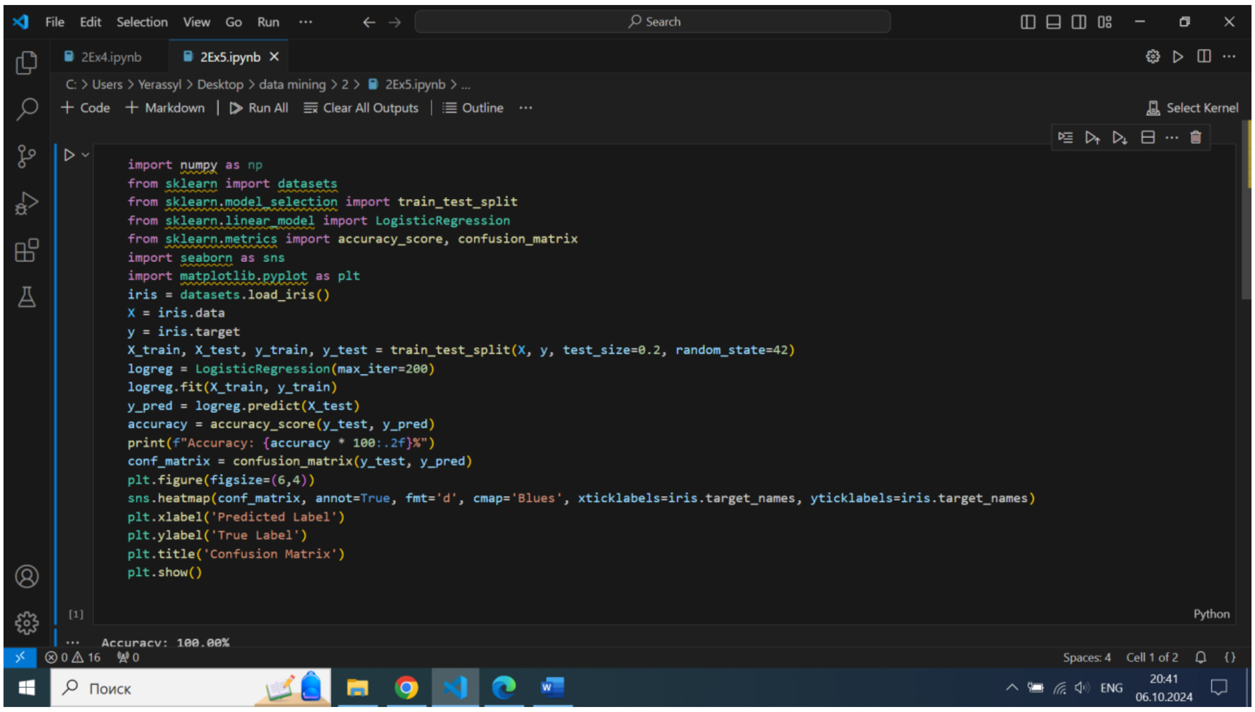
Exercise 1: Logistic Regression

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.

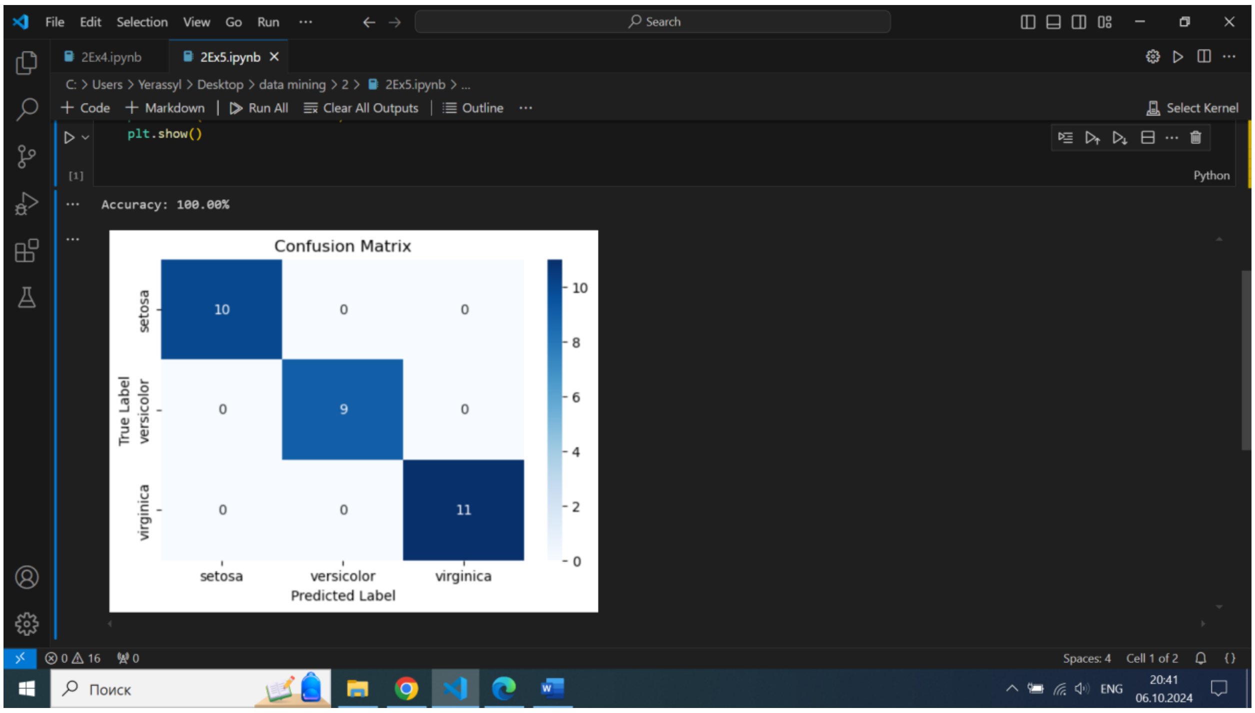
4. Evaluate the model's performance on the test set using accuracy and a confusion matrix.

We build the logistic regression model with the previous Iris dataset.



```
import numpy as np
from sklearn import datasets
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 = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, 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 * 100:.2f}%")
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 100.00%
```



```
plt.show()
```

```
Accuracy: 100.00%
```

Steps:

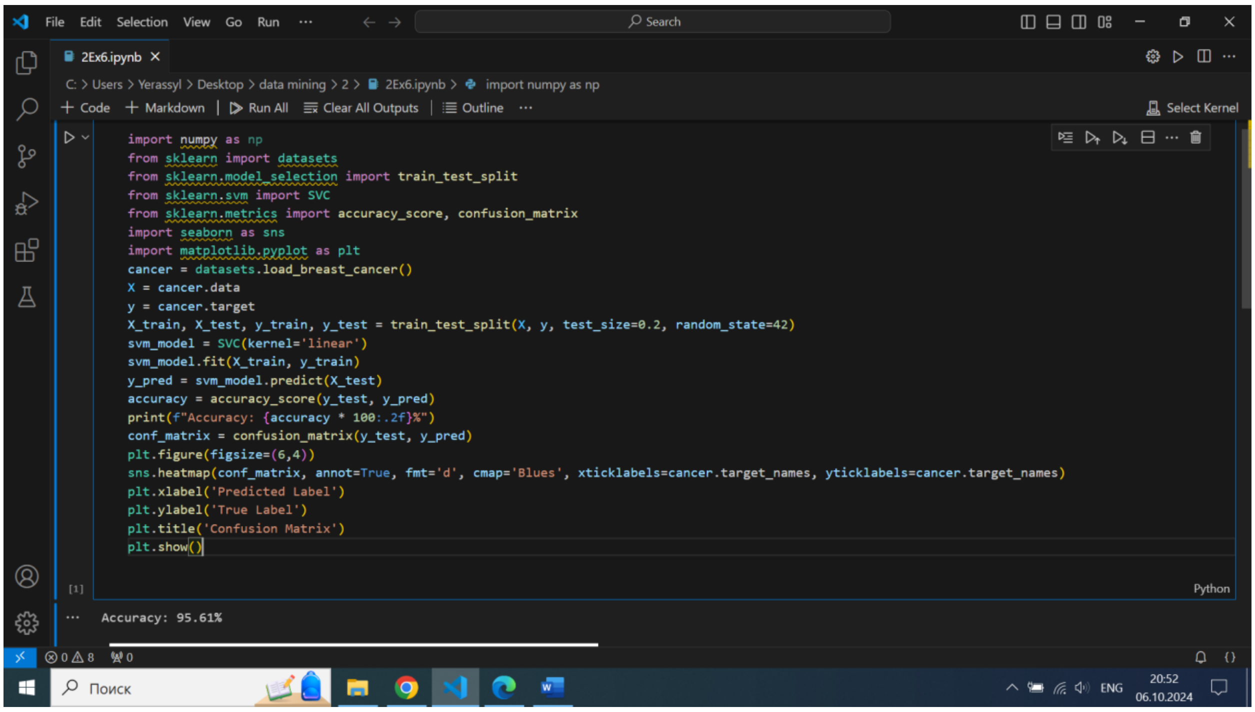
1) Using the new libraries as seaborn to visualize and accuracy_score, confusion_matrix for evaluating model's performance.
2) **logreg = LogisticRegression(max_iter=200) logreg.fit(X_train, y_train)** codes show that logistic regression works with maximum 200 iterations and another code train the data.
3) Making predictions and calculate the accuracy by the functions.
4) Display the cofussion matrix.

Exercise 2: Support Vector Machine (SVM)

Objective: Use an SVM classifier to classify data.

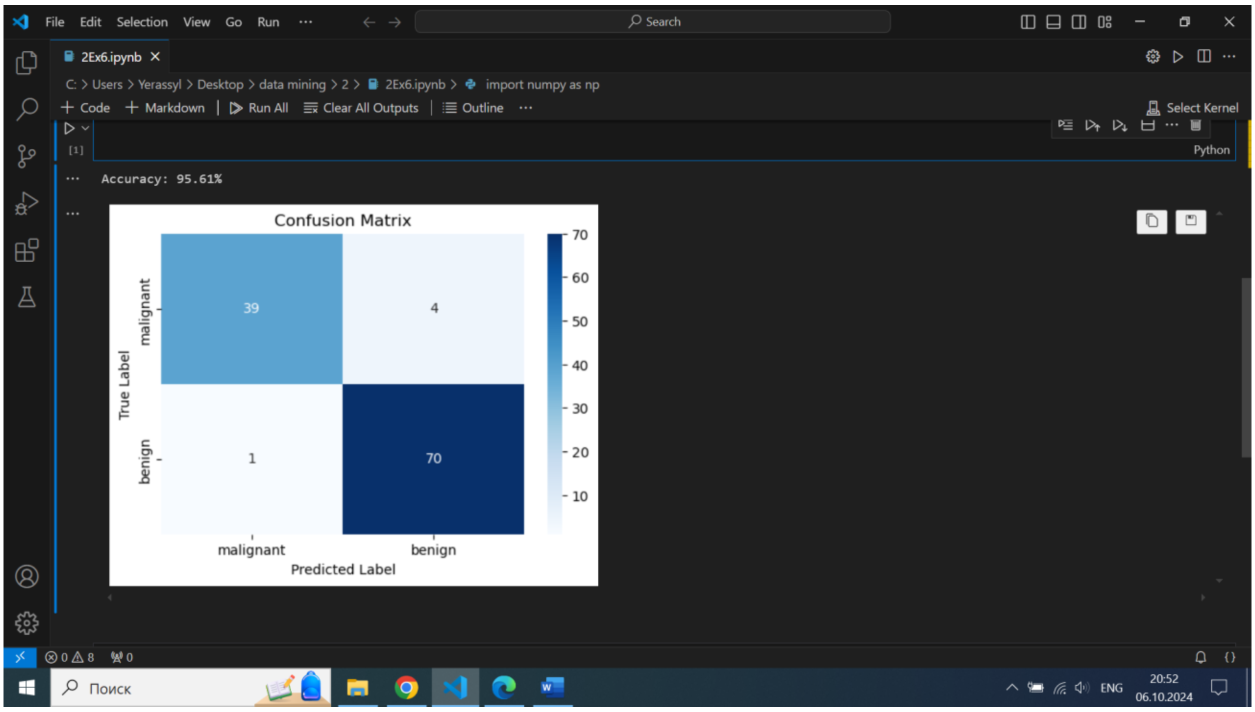
1. Load the Breast Cancer dataset from scikit-learn.
2. Split the dataset into training and testing sets.
3. Train an SVM model on the training data.
4. Evaluate the model's performance on the test data using accuracy and a confusion matrix.

Using the SVM classification for Cancer dataset.


File Edit Selection View Go Run ···
Search
2Ex6.ipynb
C: > Users > Yerassyl > Desktop > data mining > 2 > 2Ex6.ipynb > import numpy as np
+ Code + Markdown | Run All Clear All Outputs | Outline ···
Select Kernel
Python
Поиск
ENG
20:52
06.10.2024

```
import numpy as np
from sklearn import datasets
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
cancer = datasets.load_breast_cancer()
X = cancer.data
y = cancer.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, 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 * 100:.2f}%")
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=cancer.target_names, yticklabels=cancer.target_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
[1]
Accuracy: 95.61%
```



```
Accuracy: 95.61%
```

Steps:

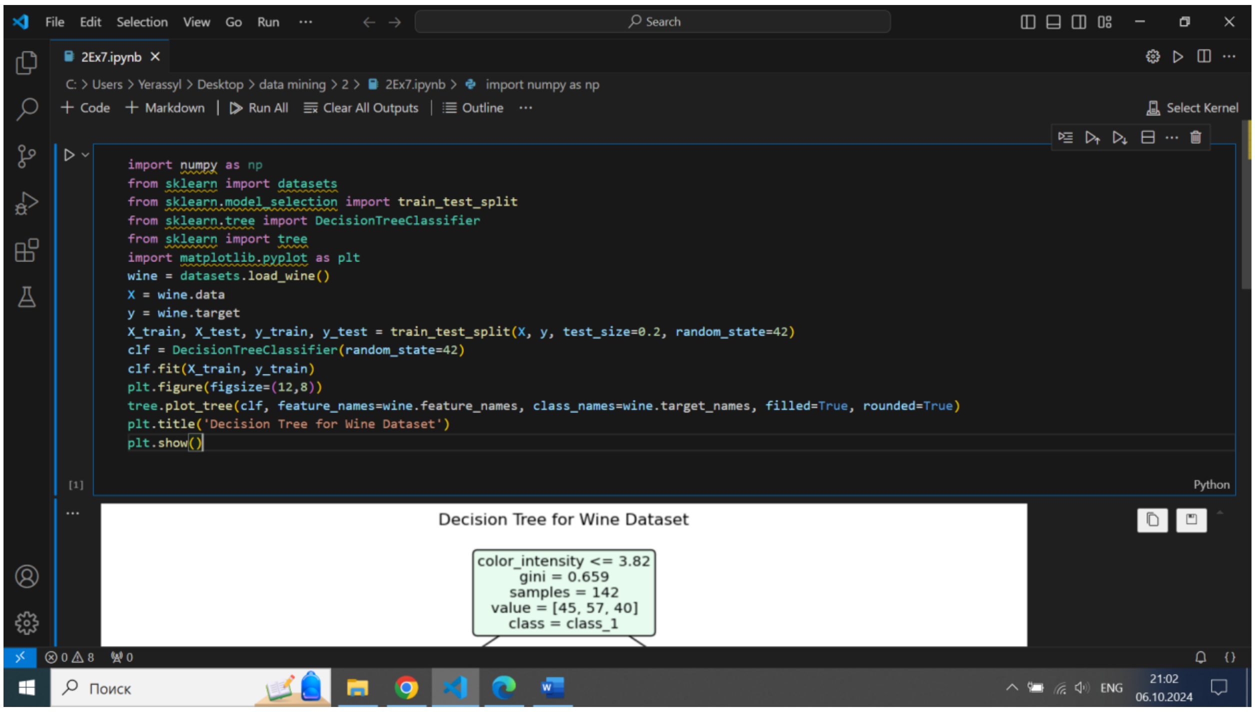
All steps as the previous task but there we have accuracy 95.61 percents. We got this result by dividing the 109/114=0.9561, the number of unpredicted data is 5.

Exercise 3: Decision Tree Classifier

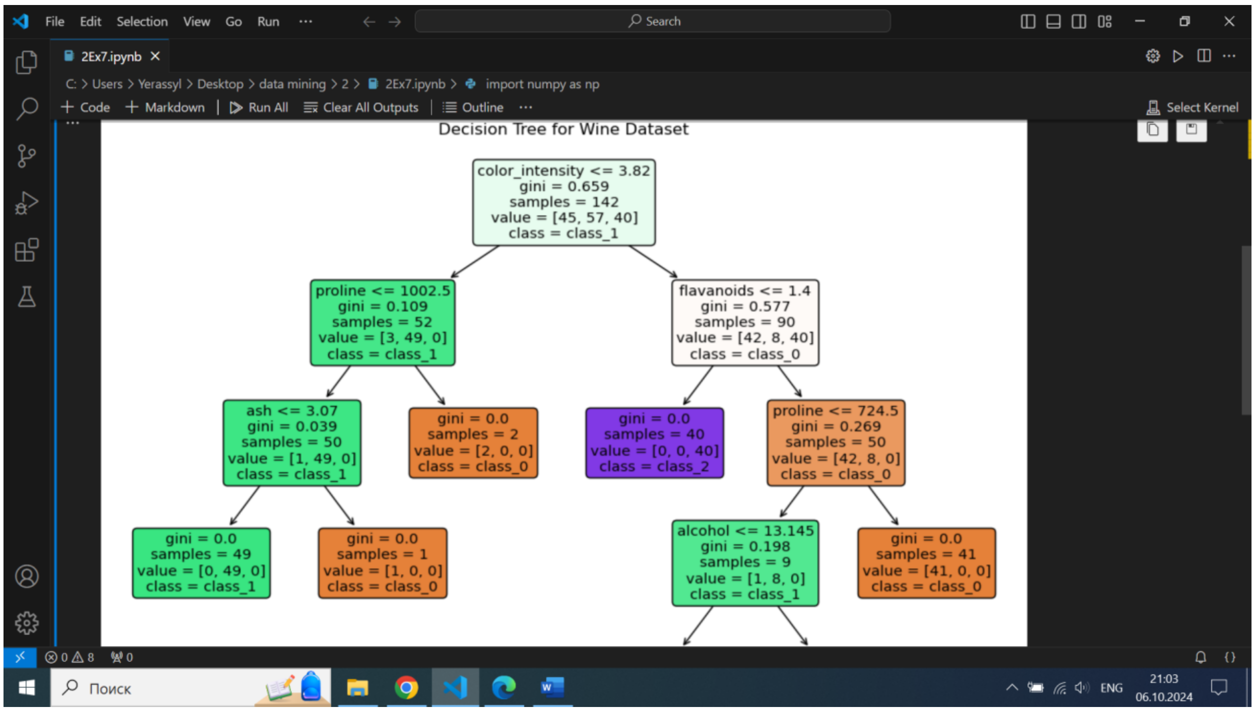
Objective: Build a decision tree classifier and visualize it.

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.

Task show the visual classification of Wine dataset.


File Edit Selection View Go Run
2Ex7.ipynb
C: > Users > Yerassyl > Desktop > data mining > 2 > 2Ex7.ipynb > import numpy as np
Code
Markdown
Run All
Clear All Outputs
Outline
Select Kernel
Python
Поиск
21:02
06.10.2024

```python
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
wine = datasets.load_wine()
X = wine.data
y = wine.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
plt.figure(figsize=(12,8))
tree.plot_tree(clf, feature_names=wine.feature_names, class_names=wine.target_names, filled=True, rounded=True)
plt.title('Decision Tree for Wine Dataset')
plt.show()
```



Steps:

The task visualize the tree of dataset by **the tree.plot_tree(clf, feature_names=wine.feature_names, class_names=wine.target_names, filled=True, rounded=True)** part of code.

Regression Exercises

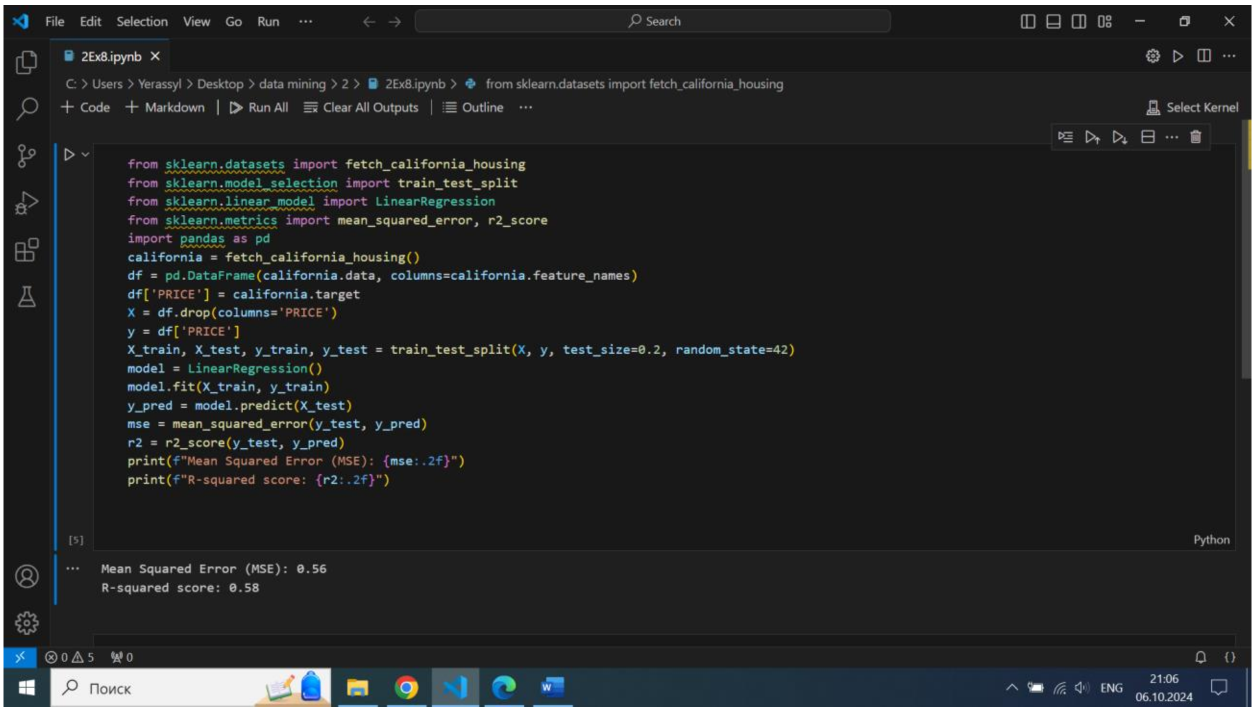
Exercise 1: Linear Regression

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.

In this task we will predict the continuous target variable by building the linear regression model. We had troubles with The Boston Housing dataset and decided to change to the California housing dataset.



```python
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
import pandas as pd
california = fetch_california_housing()
df = pd.DataFrame(california.data, columns=california.feature_names)
df['PRICE'] = california.target
X = df.drop(columns='PRICE')
y = df['PRICE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 0.56
R-squared score: 0.58
```

Steps:

1) Using libraries as previously. We have r2_score that as mean_squared_error evaluates the model performance.
2) Loading and splitting dataset.
3) Train and make the predictions.
4) Evaluate the model performance. **Mean Squared Error (MSE):** This metric measures the average squared difference between actual and predicted values. Lower MSE indicates better model performance**. R-squared Score:** This metric indicates how well the model explains the variability of the target variable. It ranges from 0 to 1, where 1 indicates that the model perfectly explains the variability.
5) Get the results.

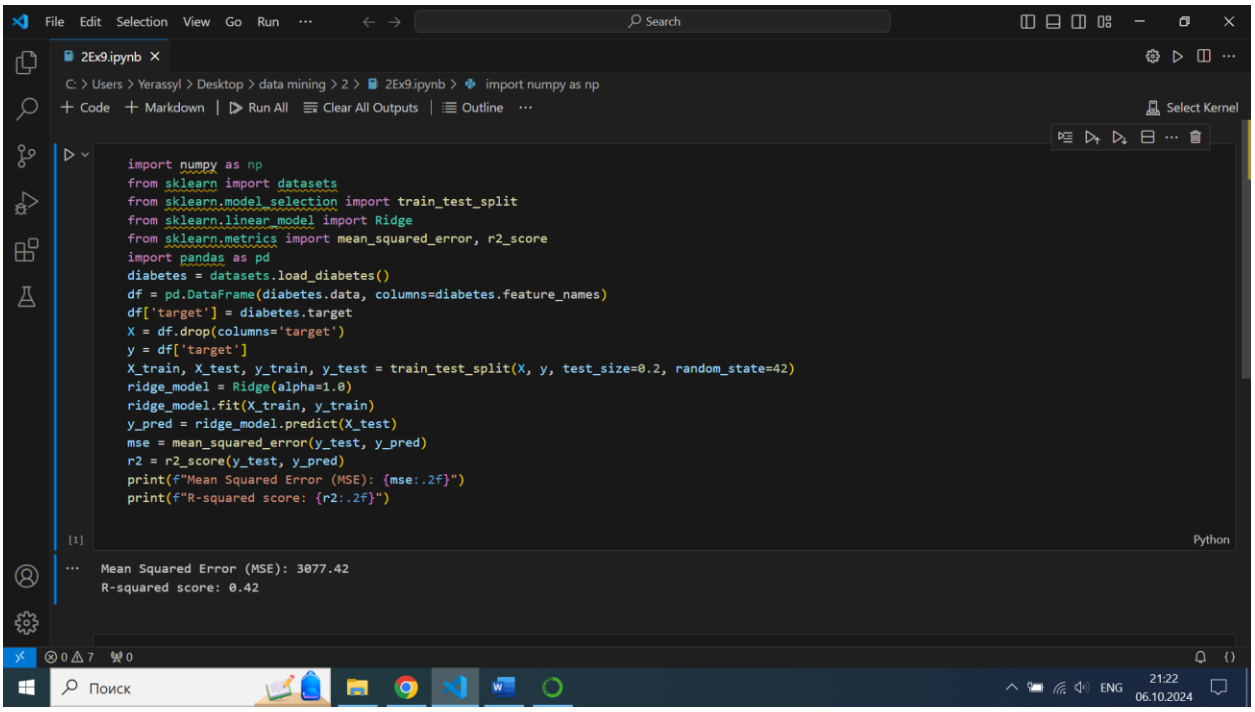
Exercise 2: Ridge Regression

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.

In this task we use ridge regression to perform linear regression.



```python
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
diabetes = datasets.load_diabetes()
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
df['target'] = diabetes.target
X = df.drop(columns='target')
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
y_pred = ridge_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 3077.42
R-squared score: 0.42
```

Steps:

1)We use new Ridge library.

2)Load the dataset and split it.

3) ridge_model = Ridge(alpha=1.0) and ridge_model.fit(X_train, y_train) codes set the ridge and train. Ridge regression is similar to linear regression but includes L2 regularization, which adds a penalty to the magnitude of the coefficients to prevent overfitting.

4) Making predictions and evaluate the models.

5) In the result, we Got the result MSE:3077.42 and R-squared score:0.42

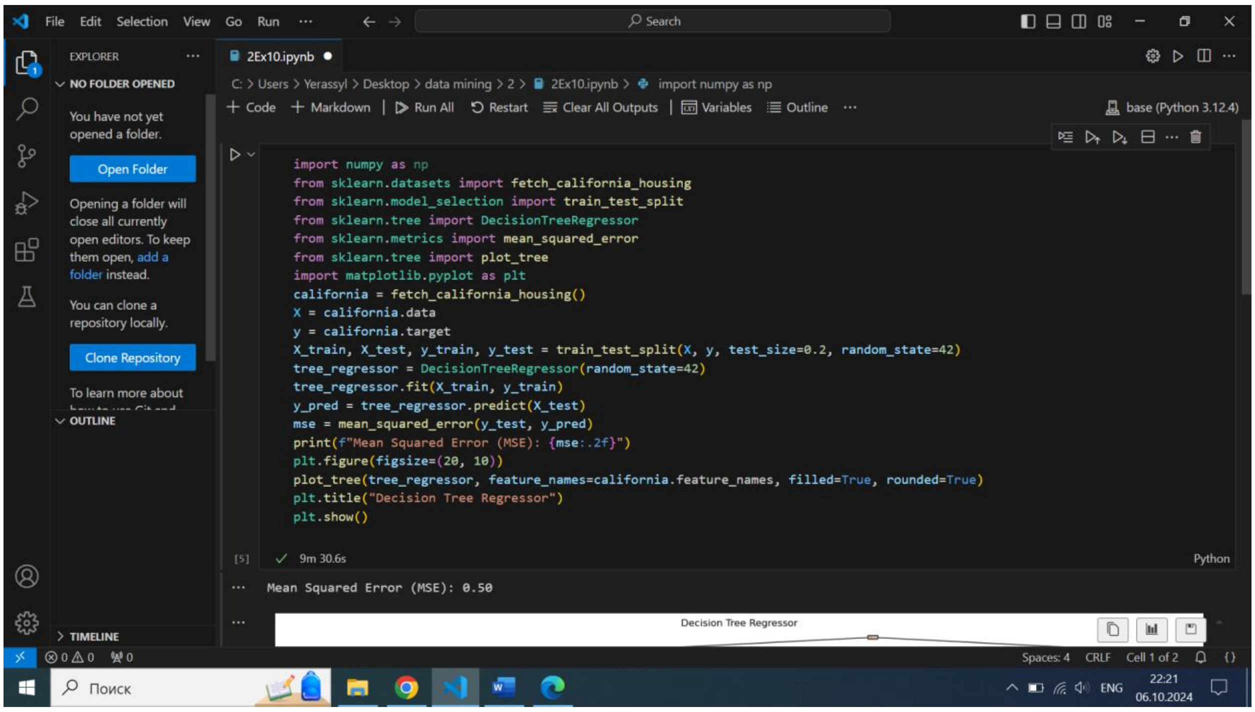
Exercise 3: Decision Tree Regression

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.

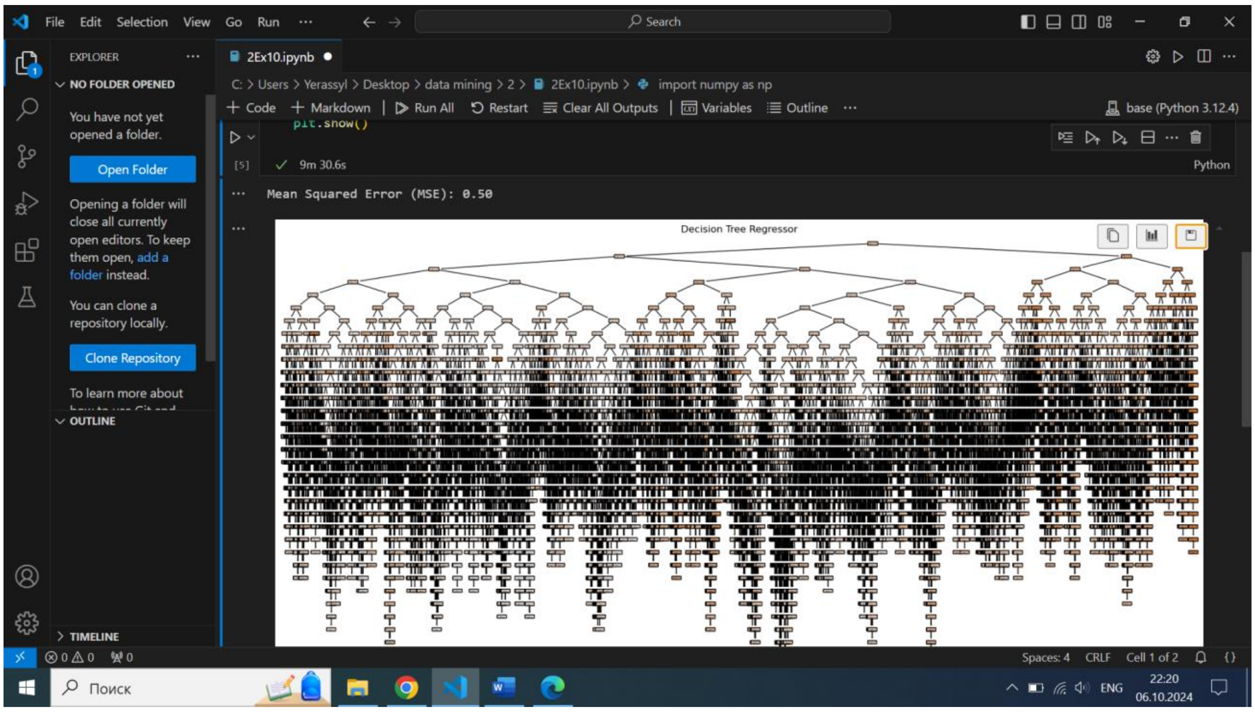
We build a tress regression model and visualize. We also used California housing because of version on scikit.


2Ex10.ipynb
C: > Users > Yerassyl > Desktop > data mining > 2 > 2Ex10.ipynb > import numpy as np
base (Python 3.12.4)
Decision Tree Regressor

```python
import numpy as np
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.tree import plot_tree
import matplotlib.pyplot as plt
california = fetch_california_housing()
X = california.data
y = california.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
tree_regressor = DecisionTreeRegressor(random_state=42)
tree_regressor.fit(X_train, y_train)
y_pred = tree_regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
plt.figure(figsize=(20, 10))
plot_tree(tree_regressor, feature_names=california.feature_names, filled=True, rounded=True)
plt.title("Decision Tree Regressor")
plt.show()
```

[5] ✓ 9m 30.6s

```
Mean Squared Error (MSE): 0.50
```



[5] ✓ 9m 30.6s

```
Mean Squared Error (MSE): 0.50
```

Making summarization: By these task we could work with features, drawing the charts and worked with new libraries and algorithms of ml.