MIT ART DESIGN & TECHNOLOGY UNIVERSITY

MIT College of Management (MITCOM), Pune

LAB -MANUAL

On

DATA MODELING AND EVALUTION

MCA (Data Sci. & Cloud Computing)

Subject Code: 21MCDS403

SEMESTER -IV (2023-25)

List of Lab Experiments

(Index)

Sr.No.	Topic	Page No.	Date	Sign.
1	Data Model Design, ER Model			
2	Data Preparation and Feature Engineering			
3	Linear Regression and Its Applications			
4	Decision Trees and Random Forest			
5	Tree-Based Models			
6	Gradient Boosting Machines (GBM, XGBoost, LightGBM)			
7	Clustering (K-Means, Hierarchical Clustering) and Dimensionality Reduction (PCA, t-SNE)			
8	Classification Metrics: Confusion Matrix, Accuracy, Precision, Recall, F1 Score			
9	Time Series Modeling ARIMA, SARIMA, and Prophet			
10	Model Validation Techniques			
11	Overfitting and Underfitting			

Experiment 1: **Data Model Design, ER Model**

Experiment -:

1. Normalize a given dataset up to 3NF.
2. Perform denormalization on a normalized dataset and compare results.
3. Write the ER-Diagram of Online Shopping Management System.
4. Convert the Table into First Normal Form (1NF).
5. Drow the ER- Diagram of University Management System.

Consider the following unnormalized table (**UNF**) for a **Student Course Registration System**:

Student_ID	Student_Name	Course_ID	Course_Name	Instructor	Instructor_Phone
101	Alice	CSE101	Database	Prof. John	123-456-7890
101	Alice	CSE102	Algorithms	Prof. Smith	987-654-3210
102	Bob	CSE101	Database	Prof. John	123-456-7890

Answer -:

1. Normalize the given dataset up to 3NF

Unnormalized Tab:

Student_ID	Student_Name	Course_ID	Course_Name	Instructor	Instructor_Phone
101	Alice	CSE101	Database	Prof. John	123-456-7890
101	Alice	CSE102	Algorithms	Prof. Smith	987-654-3210
101	Bob	CSE101	Database	Prof. John	123-456-7890

→ 1NF (Atomic Values):
All values are already atomic; table is in 1NF.

→ 2NF (Remove Partial Dependencies): Split into:

Student Table:
Student_ID | Student_Name

Course Table:
Course_ID | Course_Name | Instructor | Instructor_Phone

Registration Table:
Student_ID | Course_ID

→ 3NF (Remove Transitive Dependencies):

Instructor Table:
Instructor | Instructor_Phone

Now split:

Course Table:
Course_ID | Course_Name | Instructor

Final Tables in 3NF:

1. **Student:**
   Student_ID | Student_Name
2. **Course:**
   Course_ID | Course_Name | Instructor
3. **Instructor:**
   Instructor | Instructor_Phone
4. **Registration:**
   Student_ID | Course_ID

2. Denormalization & Comparison

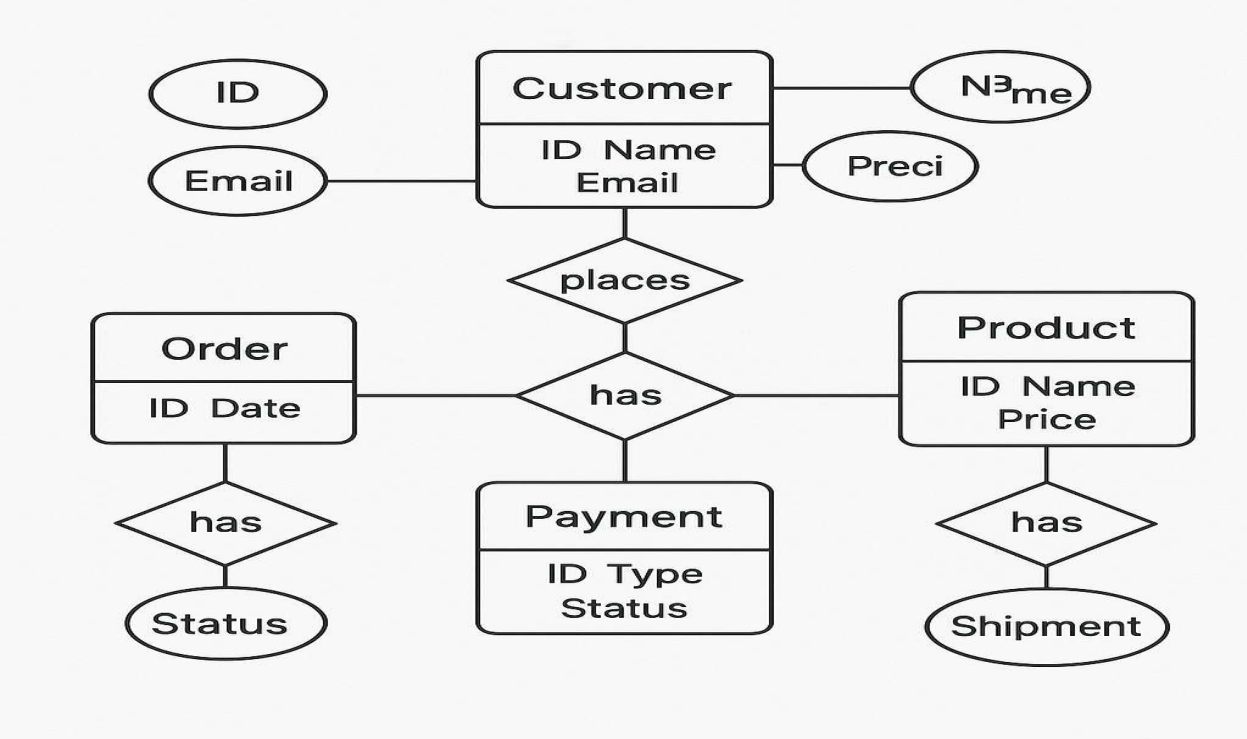
Denormalized Table:

Merge all into one:

Student_ID	Student_Name	Course_ID	Course_Name	Instructor	Instructor_Phone

Criteria	**Normalized**	**Denormalized**
Redundancy	Reduced	High (Instructor details repeated)
Storage	Efficient	More space used
Query Speed	More joins, slower	Faster read (fewer joins)
Update Anomaly	Avoided	Possible (e.g., phone number change)
Data Integrity	High	Low

3. ER-Diagram – Online Shopping Management System:



4. Convert to 1NF

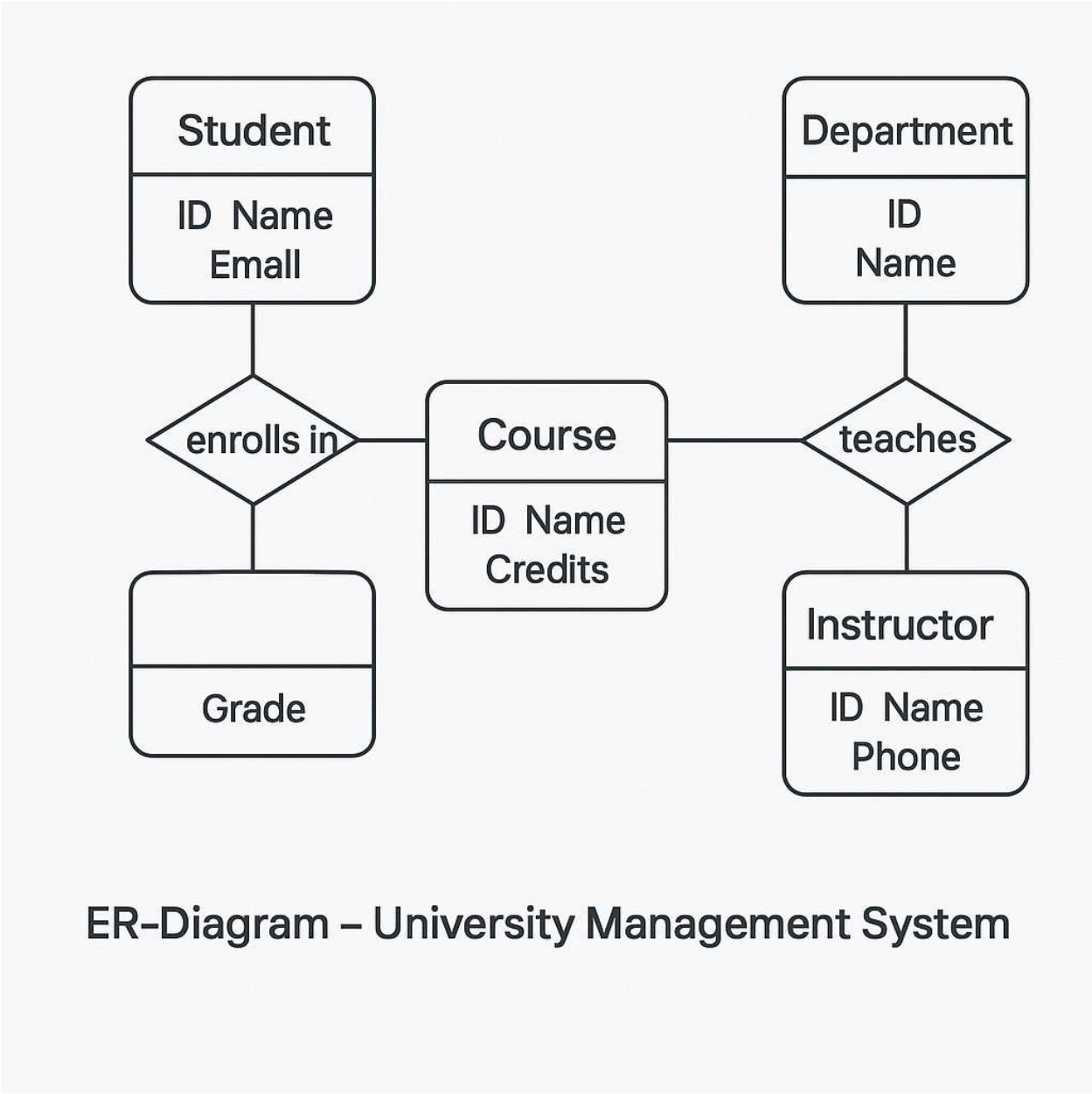
UNF Example (repeating group):

Student_ID	Name	Courses
101	Alice	CSE101, CSE102

→ Convert to 1NF (Atomic values):

Student_ID	Name	Course
101	Alice	CSE101
101	Alice	CSE102

5. ER-Diagram – University Management System:



Experiment 2-: Data Preparation and Feature Engineering

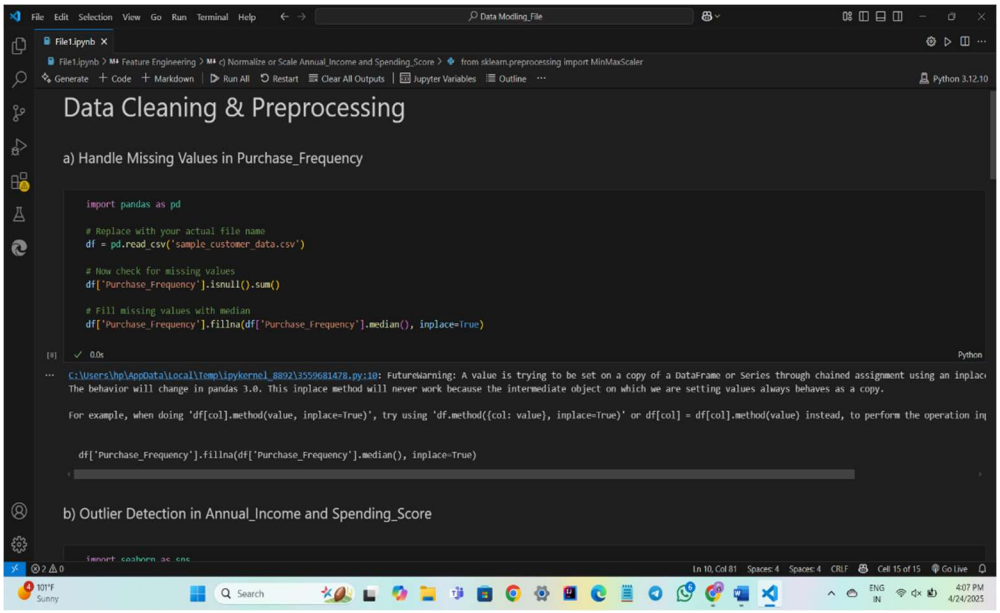
Experiment -:

1. **Data Cleaning & Preprocessing:**
   a) Identify and handle any missing values in the Purchase_Frequency column.
   b) Detect and treat potential outliers in the Annual_Income and Spending_Score columns.

2. **Encoding Categorical Variables:**
   a) Encode the Gender and Membership_Type columns appropriately for machine learning models.

3. **Feature Engineering:**
   a) Perform feature selection to identify the most relevant features for predicting spending behavior.
   b) Apply Principal Component Analysis (PCA) and explain how it helps in dimensionality reduction.
   c) Normalize or scale the Annual_Income and Spending_Score columns using an appropriate technique.

Answer -:



File Edit Selection View Go Run Terminal Help

Data Modling_File

File1.ipynb

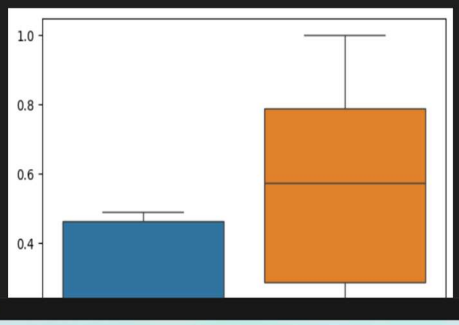
File1.ipynb > Encoding Categorical Variables

Generate + Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline ... Python 3.12.10

b) Outlier Detection in Annual_Income and Spending_Score

```
import seaborn as sns
import matplotlib.pyplot as plt

# Boxplots
sns.boxplot(data=df[['Annual_Income', 'Spending_Score']])
plt.show()

# Treat outliers using IQR
for col in ['Annual_Income', 'Spending_Score']:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 0.5 * IQR
    upper_bound = Q3 + 0.5 * IQR
    df[col] = df[col].clip(lower=lower_bound, upper=upper_bound)
```

[18] ✓ 0.1s Python



⊗ 2 △ 0 Ln 10, Col 81 Spaces: 4 Spaces: 4 CRLF Cell 6 of 15 Go Live

101°F Sunny Search ENG IN 4:10 PM 4/24/2025

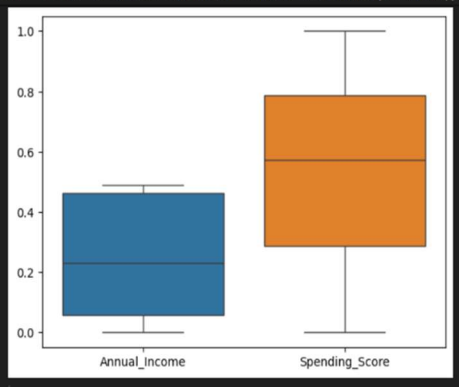
File Edit Selection View Go Run Terminal Help

Data Modling_File

File1.ipynb

File1.ipynb > Encoding Categorical Variables

Generate + Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline ... Python 3.12.10



Encoding Categorical Variables

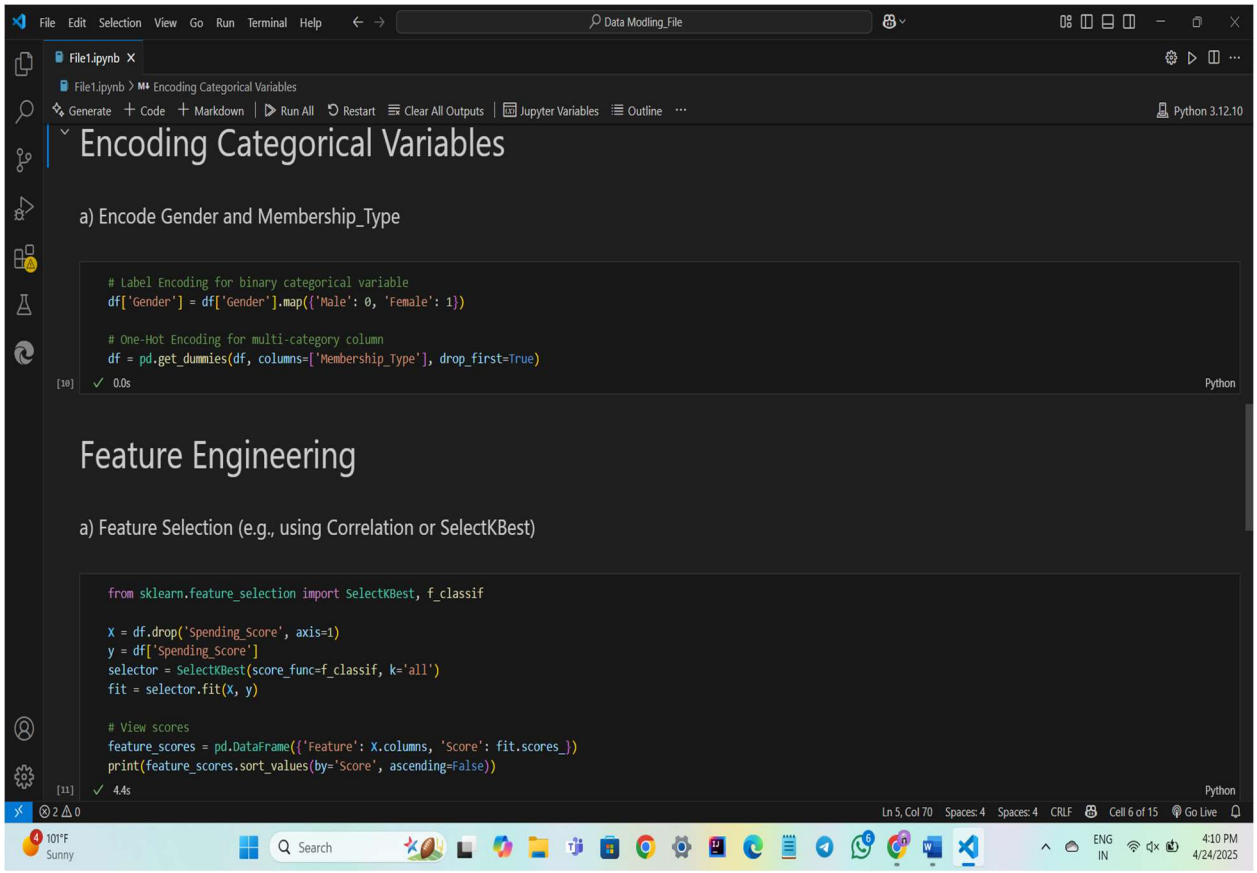
a) Encode Gender and Membership_Type

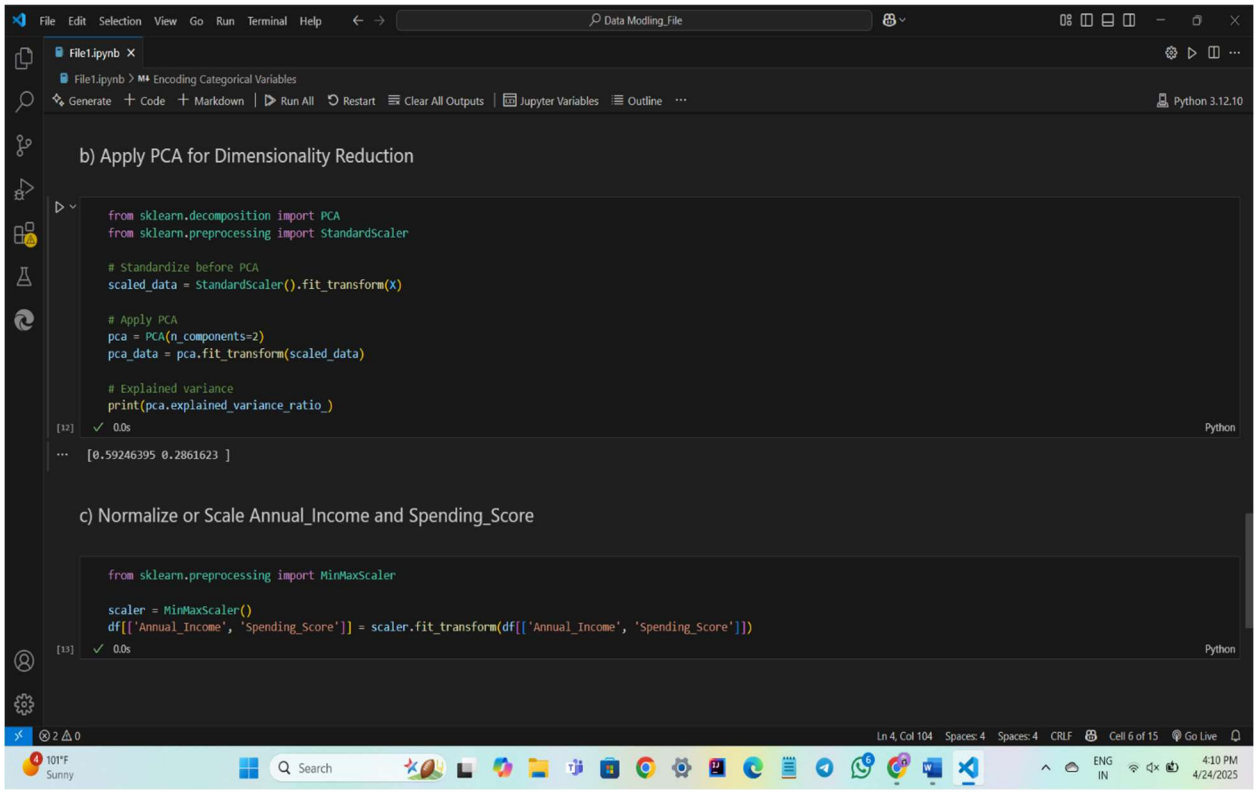
```
# Label Encoding for binary categorical variable
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
```

[10]

⊗ 2 △ 0 Ln 5, Col 70 Spaces: 4 Spaces: 4 CRLF Cell 6 of 15 Go Live

101°F Sunny Search ENG IN 4:10 PM 4/24/2025


File Edit Selection View Go Run Terminal Help
Data Modling_File
File1.ipynb
File1.ipynb > Encoding Categorical Variables
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.10
Encoding Categorical Variables
a) Encode Gender and Membership_Type
Label Encoding for binary categorical variable
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
# One-Hot Encoding for multi-category column
df = pd.get_dummies(df, columns=['Membership_Type'], drop_first=True)
[10] 0.0s
Python
Feature Engineering
a) Feature Selection (e.g., using Correlation or SelectKBest)
from sklearn.feature_selection import SelectKBest, f_classif
X = df.drop('Spending_Score', axis=1)
y = df['Spending_Score']
selector = SelectKBest(score_func=f_classif, k='all')
fit = selector.fit(X, y)
# View scores
feature_scores = pd.DataFrame({'Feature': X.columns, 'Score': fit.scores_})
print(feature_scores.sort_values(by='Score', ascending=False))
[11] 4.4s
Python
Ln 5, Col 70 Spaces: 4 Spaces: 4 CRLF Cell 6 of 15 Go Live
101°F Sunny
Search
ENG IN
4:10 PM 4/24/2025

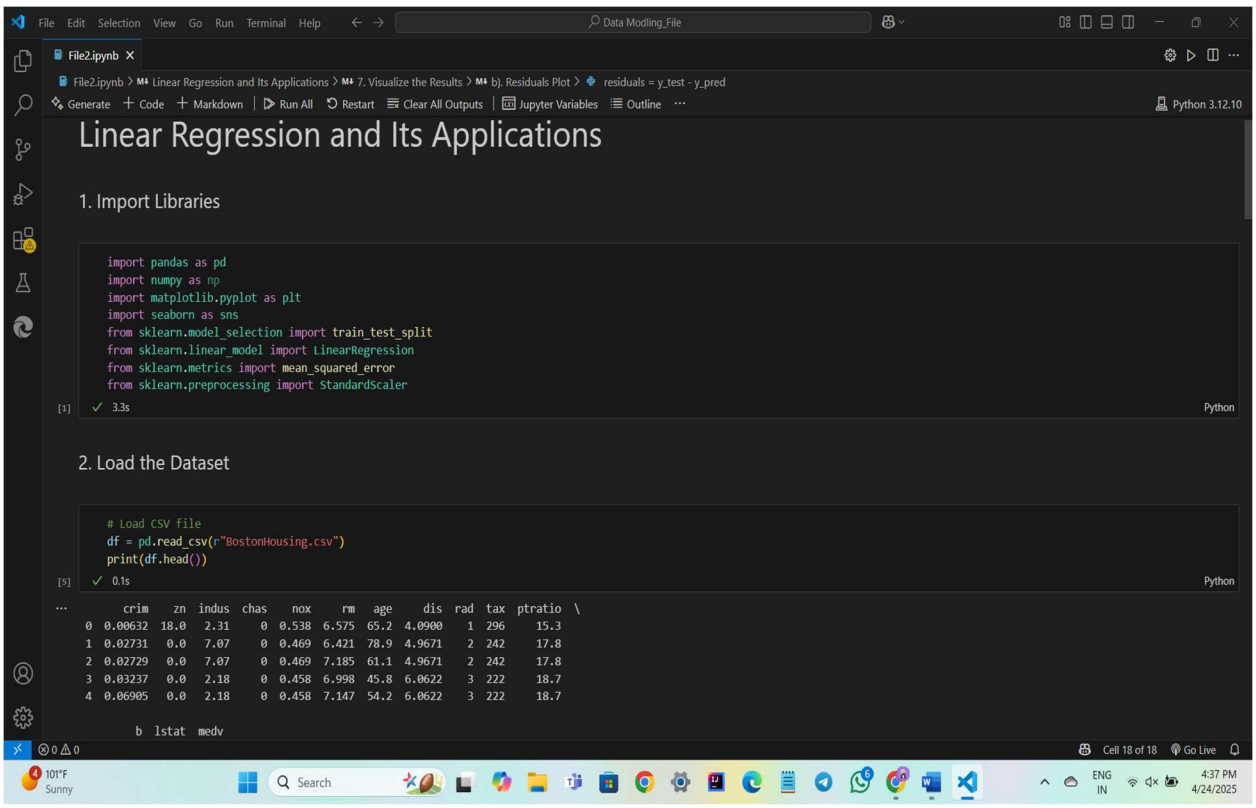

File Edit Selection View Go Run Terminal Help
Data Modling_File
File1.ipynb
File1.ipynb > Encoding Categorical Variables
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.10
b) Apply PCA for Dimensionality Reduction
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Standardize before PCA
scaled_data = StandardScaler().fit_transform(X)
# Apply PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)
# Explained variance
print(pca.explained_variance_ratio_)
[12] 0.0s
Python
[0.59246395 0.2861623]
c) Normalize or Scale Annual_Income and Spending_Score
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['Annual_Income', 'Spending_Score']] = scaler.fit_transform(df[['Annual_Income', 'Spending_Score']])
[13] 0.0s
Python
Ln 4, Col 104 Spaces: 4 Spaces: 4 CRLF Cell 6 of 15 Go Live
101°F Sunny
Search
ENG IN
4:10 PM 4/24/2025

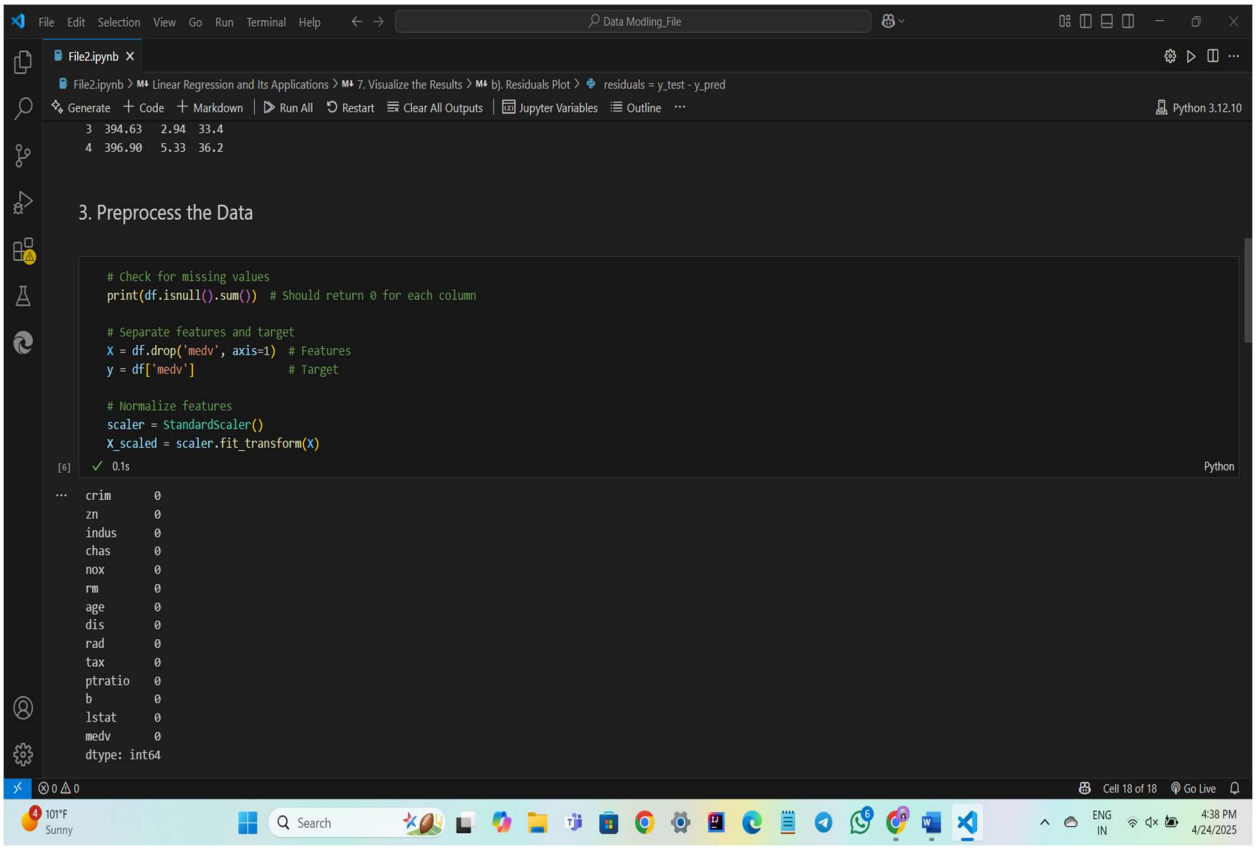
Experiment 3 -: Linear Regression and Its Applications

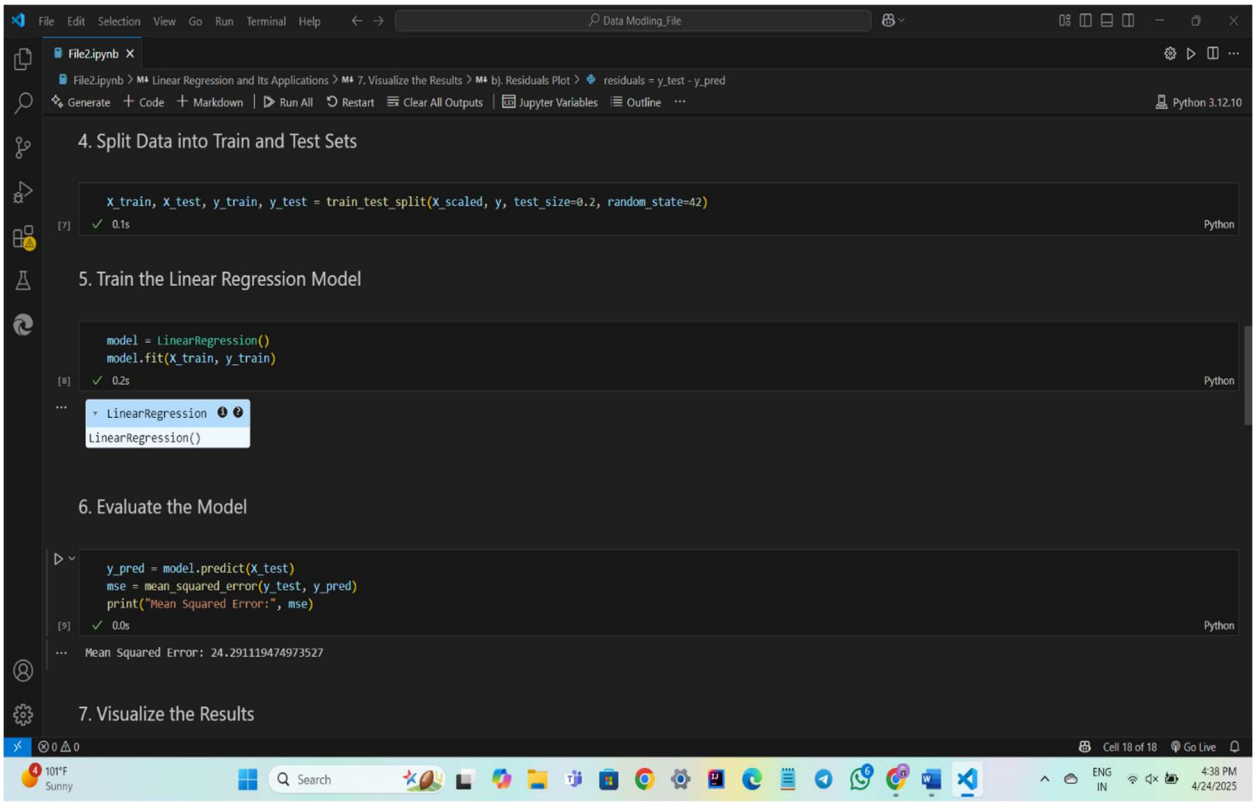
Experiment -:

1. Load a dataset (e.g., housing prices dataset).
2. Preprocess the data (handle missing values, normalize features if necessary).
3. Split the data into training and testing sets.
4. Train a linear regression model using libraries like Scikit-learn.
5. Evaluate the model using metrics like Mean Squared Error (MSE).
6. Visualize the regression line and residuals.

Answer -:



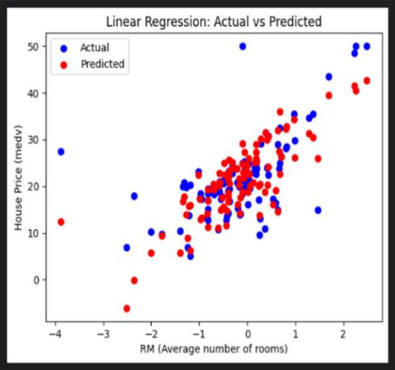

File Edit Selection View Go Run Terminal Help
Data Modling_File
File2.ipynb
File2.ipynb > Linear Regression and Its Applications > 7. Visualize the Results > b). Residuals Plot > residuals = y_test - y_pred
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.10
3 394.63 2.94 33.4
4 396.90 5.33 36.2
3. Preprocess the Data
Check for missing values
print(df.isnull().sum()) # Should return 0 for each column
# Separate features and target
X = df.drop('medv', axis=1) # Features
y = df['medv'] # Target
# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
[6] 0.1s
Python
crim 0
zn 0
indus 0
chas 0
nox 0
rm 0
age 0
dis 0
rad 0
tax 0
ptratio 0
b 0
lstat 0
medv 0
dtype: int64
Cell 18 of 18
Go Live
101°F Sunny
Search
ENG IN
4:38 PM 4/24/2025


File Edit Selection View Go Run Terminal Help
Data Modling_File
File2.ipynb
File2.ipynb > Linear Regression and Its Applications > 7. Visualize the Results > b). Residuals Plot > residuals = y_test - y_pred
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.10
4. Split Data into Train and Test Sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
[7] 0.1s
Python
5. Train the Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
[8] 0.2s
Python
LinearRegression
LinearRegression()
6. Evaluate the Model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
[9] 0.0s
Python
Mean Squared Error: 24.291119474973527
7. Visualize the Results
Cell 18 of 18
Go Live
101°F Sunny
Search
ENG IN
4:38 PM 4/24/2025

7. Visualize the Results

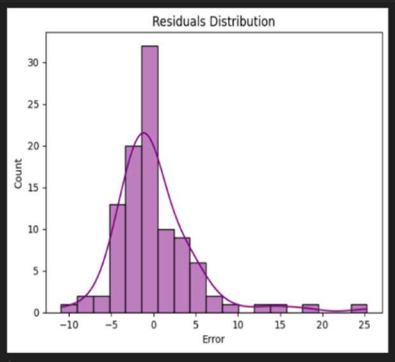
a). Regression Line (only one feature for plot)

```
# Use one feature for regression line visualization (e.g., RM: average number of rooms)
plt.scatter(X_test[:, 5], y_test, color='blue', label='Actual')
plt.scatter(X_test[:, 5], y_pred, color='red', label='Predicted')
plt.xlabel("RM (Average number of rooms)")
plt.ylabel("House Price (medv)")
plt.title("Linear Regression: Actual vs Predicted")
plt.legend()
plt.show()
```



b). Residuals Plot

```
residuals = y_test - y_pred
sns.histplot(residuals, kde=True, color='purple')
plt.title("Residuals Distribution")
plt.xlabel("Error")
plt.show()
```

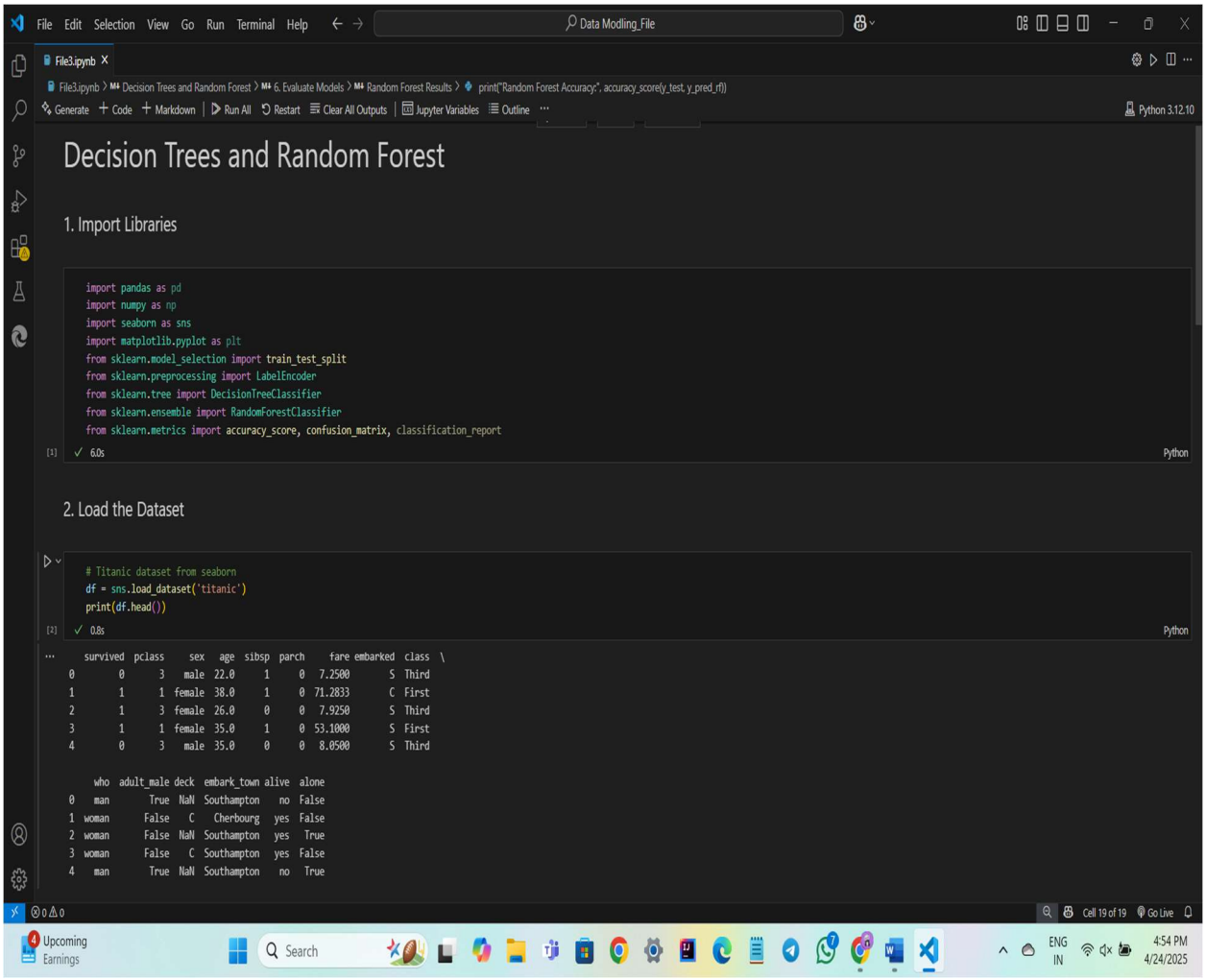


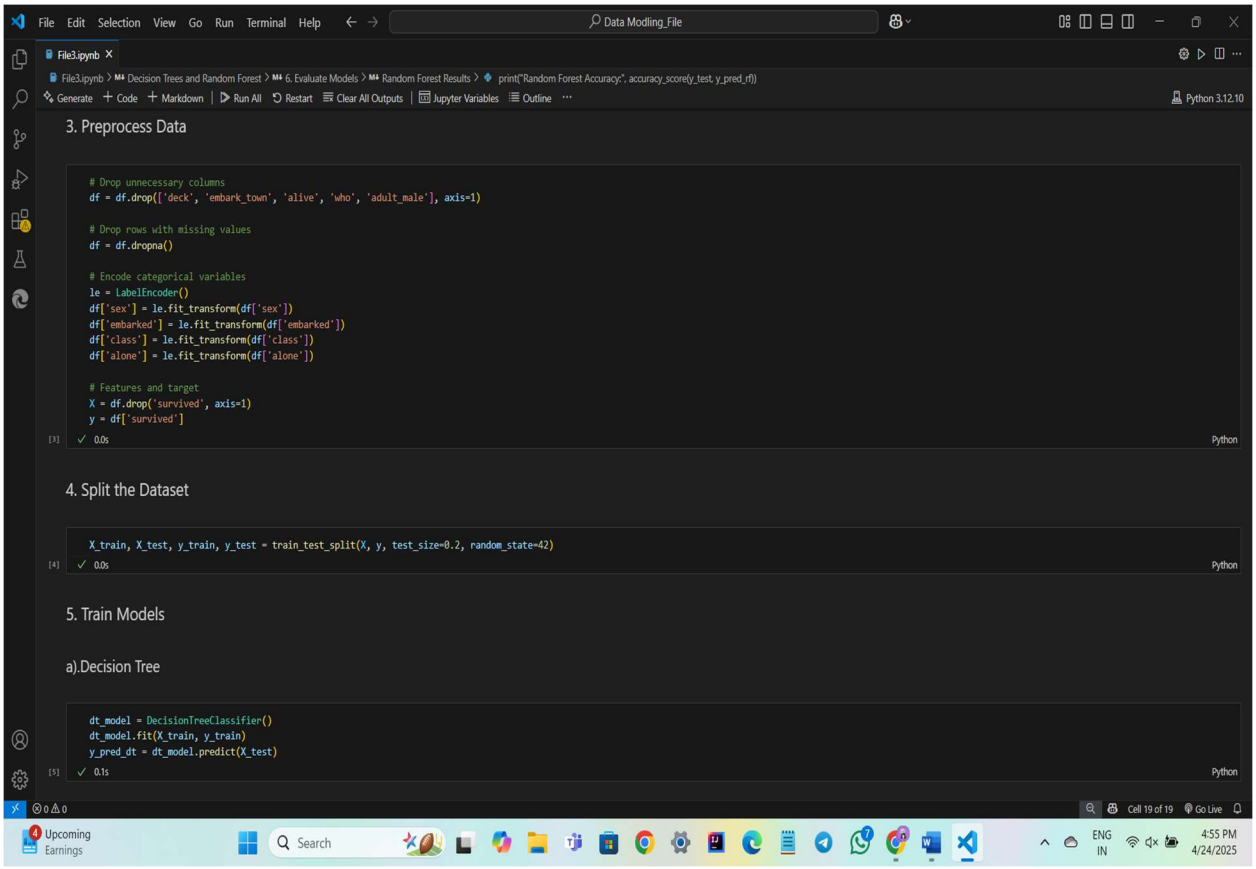
Experiment 4 -: Decision Trees and Random Forest

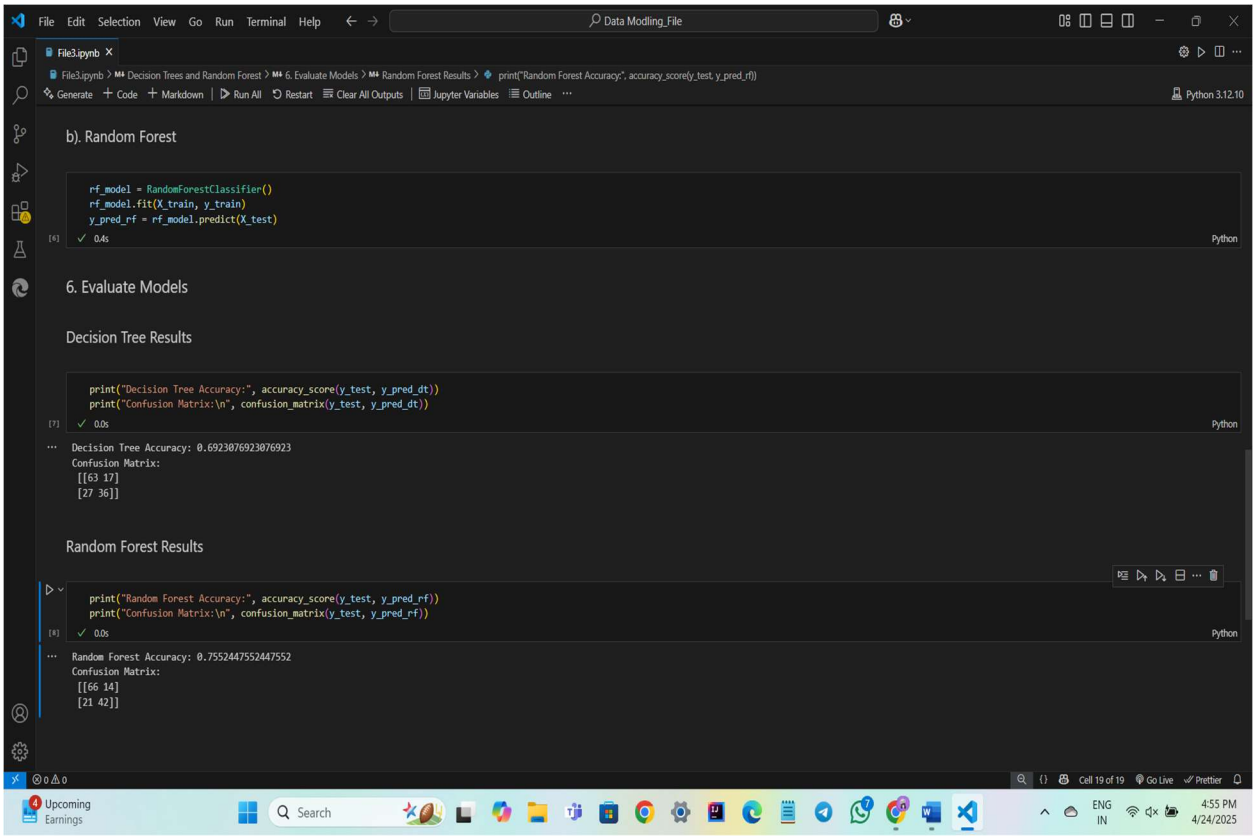
Experiment -:

1. Load a dataset (e.g., Titanic survival dataset).
2. Preprocess categorical and numerical features.
3. Split into training and testing sets.
4. Train a logistic regression model.
5. Evaluate using accuracy and confusion matrix.

Answer -:




File Edit Selection View Go Run Terminal Help
Data Modling_File
File3.ipynb
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.10
3. Preprocess Data
# Drop unnecessary columns
df = df.drop(['deck', 'embark_town', 'alive', 'who', 'adult_male'], axis=1)
# Drop rows with missing values
df = df.dropna()
# Encode categorical variables
le = LabelEncoder()
df['sex'] = le.fit_transform(df['sex'])
df['embarked'] = le.fit_transform(df['embarked'])
df['class'] = le.fit_transform(df['class'])
df['alone'] = le.fit_transform(df['alone'])
# Features and target
X = df.drop('survived', axis=1)
y = df['survived']
[3] ✓ 0.0s Python
4. Split the Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
[4] ✓ 0.0s Python
5. Train Models
a).Decision Tree
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
[5] ✓ 0.1s Python
Cell 19 of 19
Go Live
Upcoming Earnings
Search
ENG IN
4:55 PM 4/24/2025

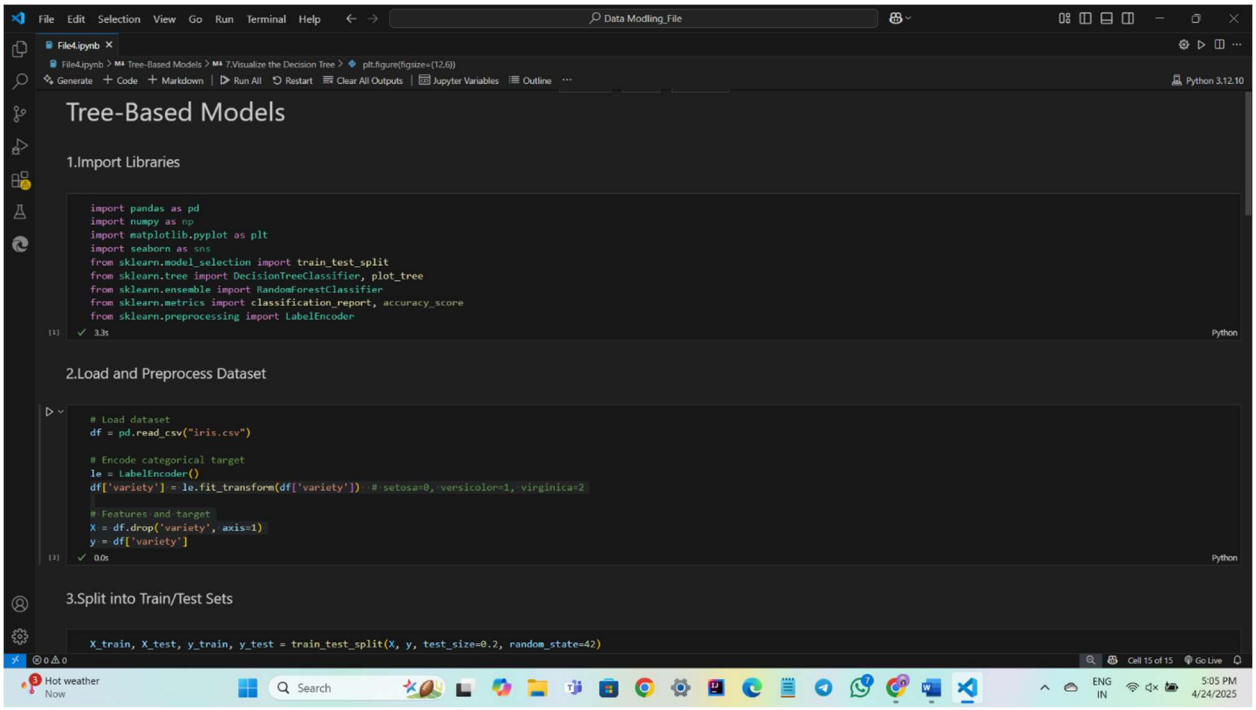

File Edit Selection View Go Run Terminal Help
Data Modling_File
File3.ipynb
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.10
b). Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
[6] ✓ 0.4s Python
6. Evaluate Models
Decision Tree Results
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
[7] ✓ 0.0s Python
Decision Tree Accuracy: 0.6923076923076923
Confusion Matrix:
[[63 17]
[27 36]]
Random Forest Results
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
[8] ✓ 0.0s Python
Random Forest Accuracy: 0.7552447552447552
Confusion Matrix:
[[66 14]
[21 42]]
Cell 19 of 19
Go Live
Prettier
Upcoming Earnings
Search
ENG IN
4:55 PM 4/24/2025

Experiment 5 -: Tree-Based Models

Experiment -:

1. Load a classification dataset.
2. Train a Decision Tree and Random Forest model.
3. Compare their accuracies.
4. Load a dataset (e.g., Titanic, Iris, or a custom dataset).
5. Preprocess data (handle missing values, encode categorical variables, and normalize if needed).
6. Split data into training and testing sets (80/20 split).
7. Train a Decision Tree classifier with different depths and criteria (Gini vs. Entropy).
8. Visualize the tree and analyze the decision paths.
9. Evaluate performance using accuracy, precision, recall, and F1-score.

Answer -:



3.Split into Train/Test Sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4.Train Decision Tree with Gini and Entropy

```
# Gini
dt_gini = DecisionTreeClassifier(criterion='gini', max_depth=3)
dt_gini.fit(X_train, y_train)
y_pred_gini = dt_gini.predict(X_test)

# Entropy
dt_entropy = DecisionTreeClassifier(criterion='entropy', max_depth=3)
dt_entropy.fit(X_train, y_train)
y_pred_entropy = dt_entropy.predict(X_test)
```

5.Train Random Forest

```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

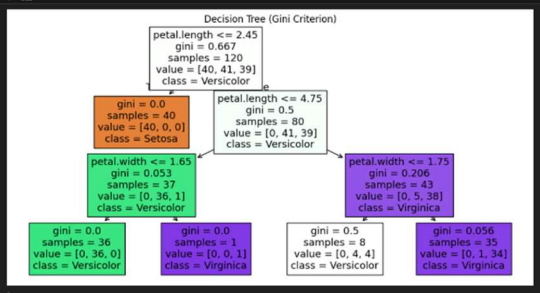
6.Evaluate Models

```
# Gini
print("Decision Tree (Gini) Report:\n", classification_report(y_test, y_pred_gini))

# Entropy
print("Decision Tree (Entropy) Report:\n", classification_report(y_test, y_pred_entropy))

# Random Forest
print("Random Forest Report:\n", classification_report(y_test, y_pred_rf))
```

```
Decision Tree (Gini) Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Decision Tree (Entropy) Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Random Forest Report:
              precision    recall  f1-score   support

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

7.Visualize the Decision Tree

```
plt.figure(figsize=(12,6))
plot_tree(dt_gini, filled=True, feature_names=X.columns, class_names=le.classes_)
plt.title("Decision Tree (Gini Criterion)")
plt.show()
```

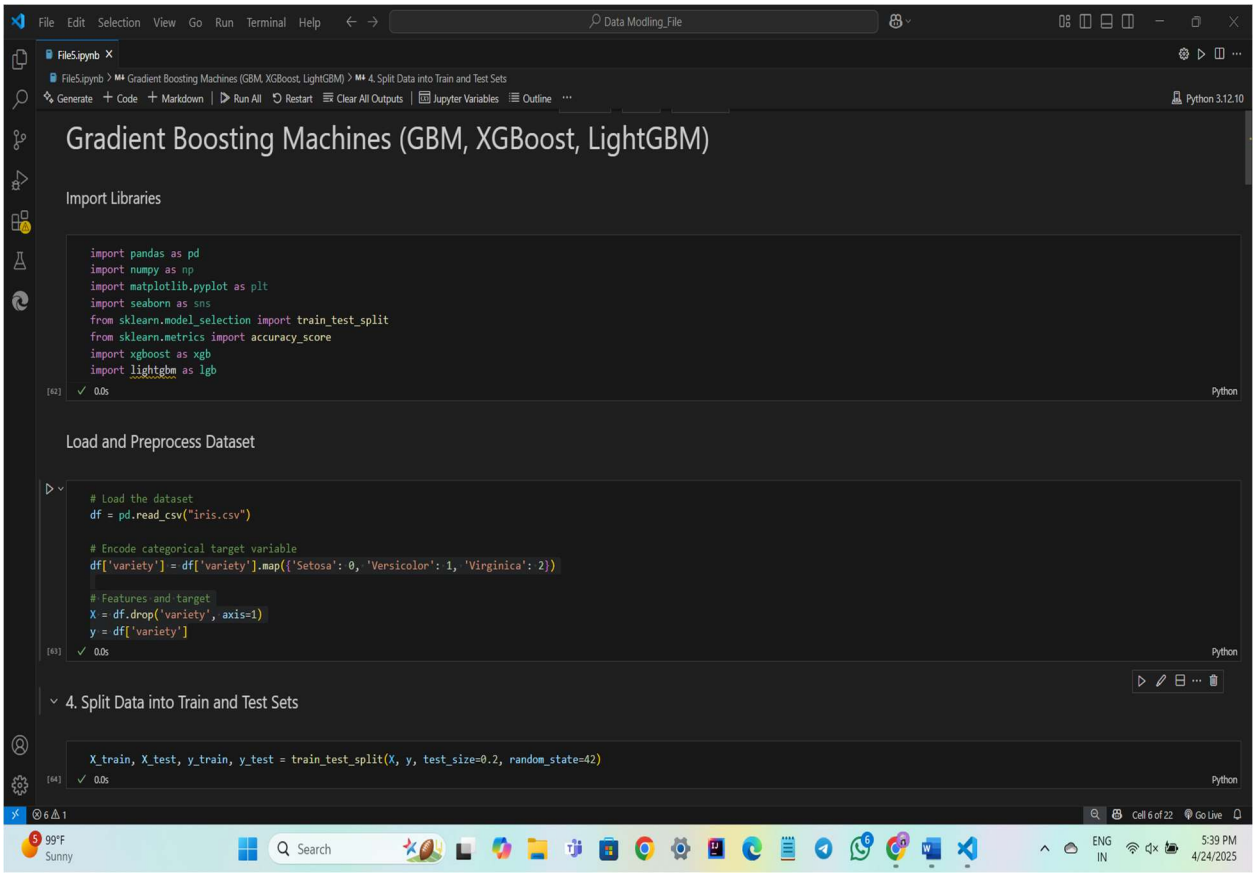


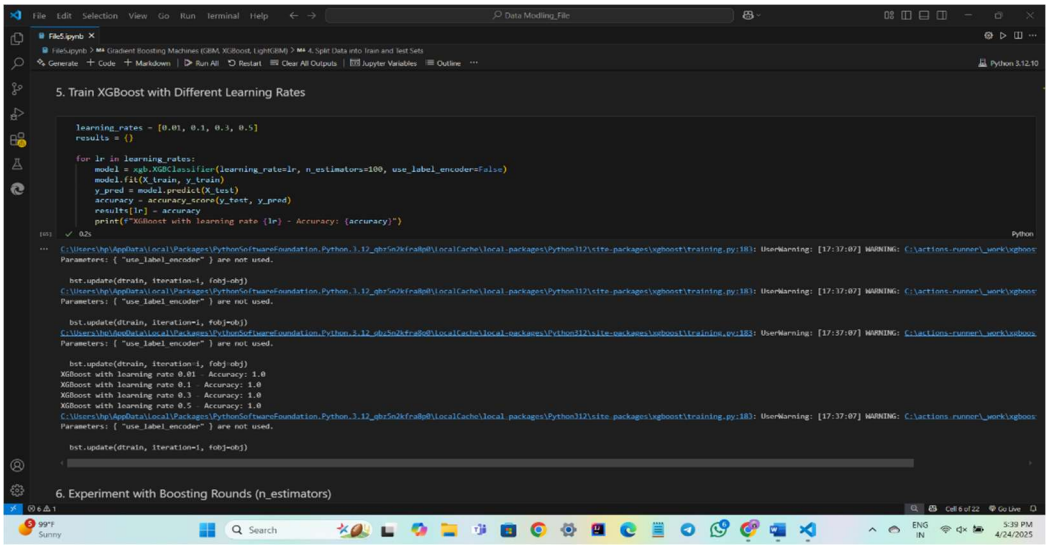
Experiment 6 -: Gradient Boosting Machines (GBM, XGBoost, LightGBM)

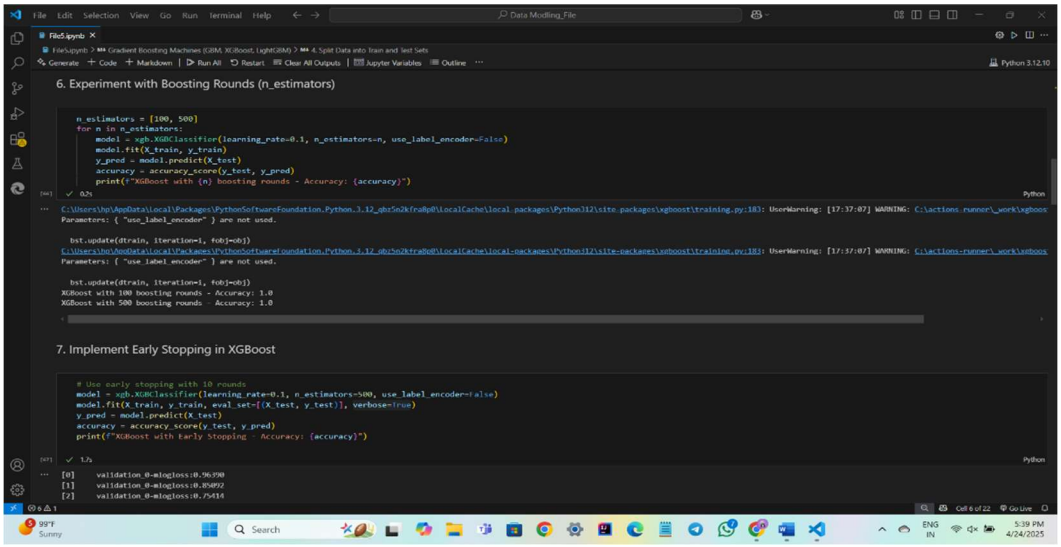
Experiment -:

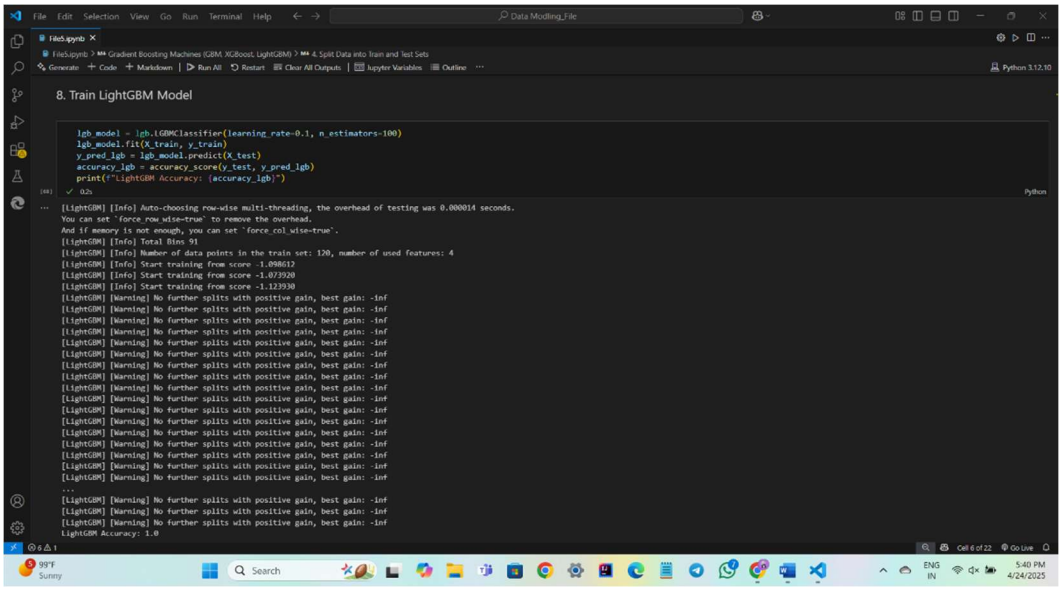
1. To implement GBM and compare with other tree models.
2. Train an XGBoost model using different values of learning rate (0.01, 0.1, 0.3, 0.5). How does accuracy change?
3. What happens if you increase the number of boosting rounds (e.g., n_estimators=500 instead of 100) in XGBoost and LightGBM?
4. Implement early stopping in XGBoost and LightGBM. How does it affect performance?
5. Try using GPU acceleration for XGBoost and LightGBM. How much does training time improve?

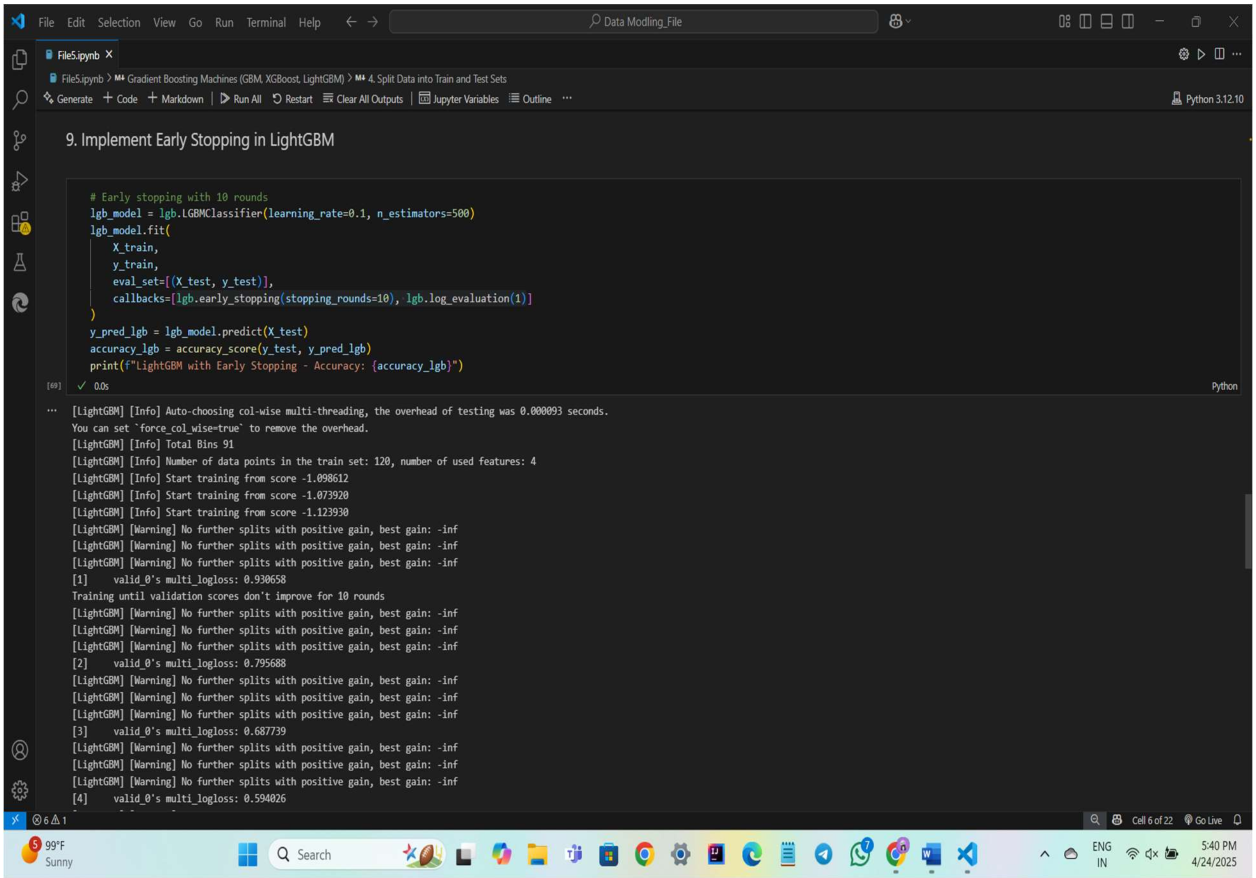
Answer -:




5. Train XGBoost with Different Learning Rates
learning_rates = [0.01, 0.1, 0.3, 0.5]
results = {}
for lr in learning_rates:
model = xgb.XGBClassifier(learning_rate=lr, n_estimators=100, use_label_encoder=False)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
results[lr] = accuracy
print(f"XGBoost with learning rate {lr} - Accuracy: {accuracy}")
XGBoost with learning rate 0.01 - Accuracy: 1.0
XGBoost with learning rate 0.1 - Accuracy: 1.0
XGBoost with learning rate 0.3 - Accuracy: 1.0
XGBoost with learning rate 0.5 - Accuracy: 1.0
6. Experiment with Boosting Rounds (n_estimators)


6. Experiment with Boosting Rounds (n_estimators)
n_estimators = [100, 500]
for n in n_estimators:
model = xgb.XGBClassifier(learning_rate=0.1, n_estimators=n, use_label_encoder=False)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"XGBoost with {n} boosting rounds - Accuracy: {accuracy}")
XGBoost with 100 boosting rounds - Accuracy: 1.0
XGBoost with 500 boosting rounds - Accuracy: 1.0
7. Implement Early Stopping in XGBoost
# Use early stopping with 10 rounds
model = xgb.XGBClassifier(learning_rate=0.1, n_estimators=500, use_label_encoder=False)
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=True)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"XGBoost with Early Stopping - Accuracy: {accuracy}")
[0] validation_0-mlogloss:0.96390
[1] validation_0-mlogloss:0.89492
[2] validation_0-mlogloss:0.75414


8. Train LightGBM Model
lgb_model = lgb.LGBMClassifier(learning_rate=0.1, n_estimators=100)
lgb_model.fit(X_train, y_train)
y_pred_lgb = lgb_model.predict(X_test)
accuracy_lgb = accuracy_score(y_test, y_pred_lgb)
print(f"LightGBM Accuracy: {accuracy_lgb}")
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000014 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 91
[LightGBM] [Info] Number of data points in the train set: 120, number of used features: 4
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.073920
[LightGBM] [Info] Start training from score -1.123930
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
LightGBM Accuracy: 1.0

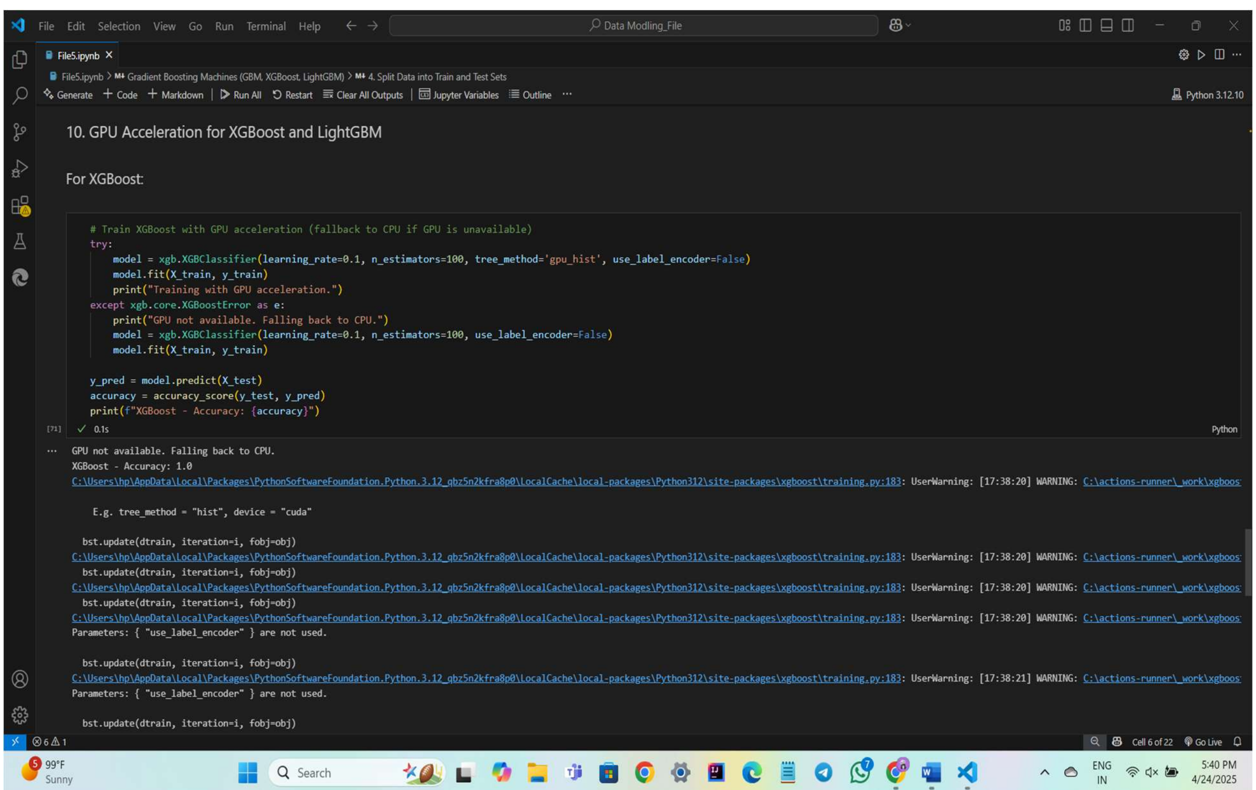


9. Implement Early Stopping in LightGBM

```python
# Early stopping with 10 rounds
lgb_model = lgb.LGBMClassifier(learning_rate=0.1, n_estimators=500)
lgb_model.fit(
    X_train,
    y_train,
    eval_set=[(X_test, y_test)],
    callbacks=[lgb.early_stopping(stopping_rounds=10), lgb.log_evaluation(1)]
)
y_pred_lgb = lgb_model.predict(X_test)
accuracy_lgb = accuracy_score(y_test, y_pred_lgb)
print(f"LightGBM with Early Stopping - Accuracy: {accuracy_lgb}")
```

[69] ✓ 0.0s Python

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000093 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 91
[LightGBM] [Info] Number of data points in the train set: 120, number of used features: 4
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.073920
[LightGBM] [Info] Start training from score -1.123930
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[1]	valid_0's multi_logloss: 0.930658
Training until validation scores don't improve for 10 rounds
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[2]	valid_0's multi_logloss: 0.795688
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[3]	valid_0's multi_logloss: 0.687739
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[4]	valid_0's multi_logloss: 0.594026
```



10. GPU Acceleration for XGBoost and LightGBM

For XGBoost:

```python
# Train XGBoost with GPU acceleration (fallback to CPU if GPU is unavailable)
try:
    model = xgb.XGBClassifier(learning_rate=0.1, n_estimators=100, tree_method='gpu_hist', use_label_encoder=False)
    model.fit(X_train, y_train)
    print("Training with GPU acceleration.")
except xgb.core.XGBoostError as e:
    print("GPU not available. Falling back to CPU.")
    model = xgb.XGBClassifier(learning_rate=0.1, n_estimators=100, use_label_encoder=False)
    model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"XGBoost - Accuracy: {accuracy}")
```

[71] ✓ 0.1s Python

```
GPU not available. Falling back to CPU.
XGBoost - Accuracy: 1.0
C:\Users\hp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\xgboost\training.py:183: UserWarning: [17:38:20] WARNING: C:\actions-runner\_work\xgboos

    E.g. tree_method = "hist", device = "cuda"

  bst.update(dtrain, iteration=i, fobj=obj)
C:\Users\hp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\xgboost\training.py:183: UserWarning: [17:38:20] WARNING: C:\actions-runner\_work\xgboos
  bst.update(dtrain, iteration=i, fobj=obj)
C:\Users\hp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\xgboost\training.py:183: UserWarning: [17:38:20] WARNING: C:\actions-runner\_work\xgboos
  bst.update(dtrain, iteration=i, fobj=obj)
C:\Users\hp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\xgboost\training.py:183: UserWarning: [17:38:20] WARNING: C:\actions-runner\_work\xgboos
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
C:\Users\hp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\xgboost\training.py:183: UserWarning: [17:38:21] WARNING: C:\actions-runner\_work\xgboos
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

Experiment 7 -: Clustering (K-Means, Hierarchical Clustering) and Dimensionality Reduction (PCA, t-SNE)

Experiment -:

Problem _ Given a dataset, apply clustering algorithms to group similar data points and use dimensionality reduction techniques for visualization and analysis.

Tasks:

1. Data Preprocessing

- Load a real-world dataset (e.g., Iris, MNIST, Wine, or any suitable dataset).
- ***Perform data cleaning and preprocessing:***
    - Handle missing values (if any).
    - Normalize or standardize the data (if needed).

2. Clustering Algorithms

- ***Apply K-Means Clustering:***
    - Use the Elbow Method to determine the optimal number of clusters.
    - Compute and visualize the clustering results.
- ***Apply Hierarchical Clustering:***
    - Use both Agglomerative and Divisive clustering.
    - Generate a dendrogram and analyze the cluster structure.

3. Dimensionality Reduction

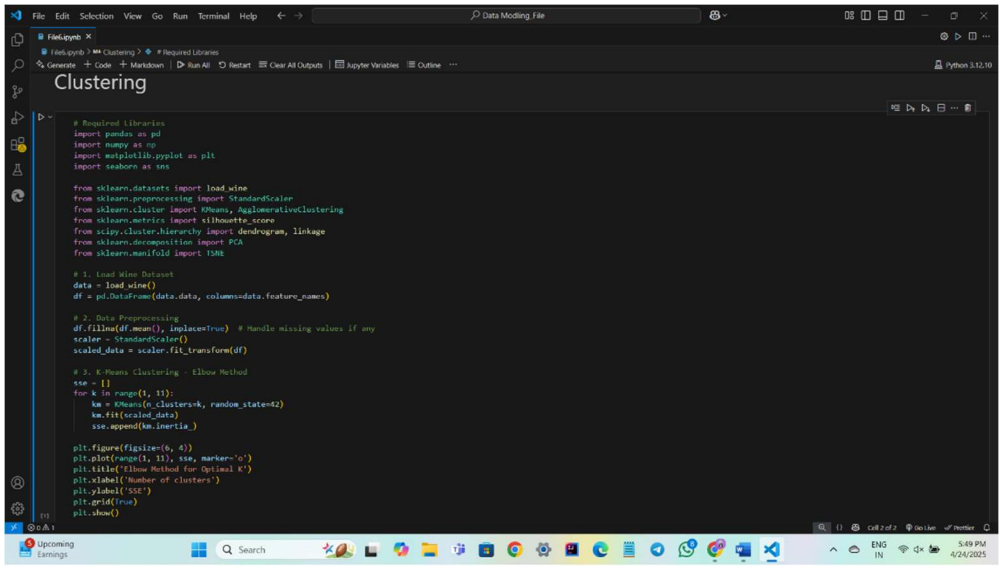
- ***Apply Principal Component Analysis (PCA):***
    - Reduce the dataset to 2 or 3 dimensions for visualization.
    - Analyze the explained variance of principal components.

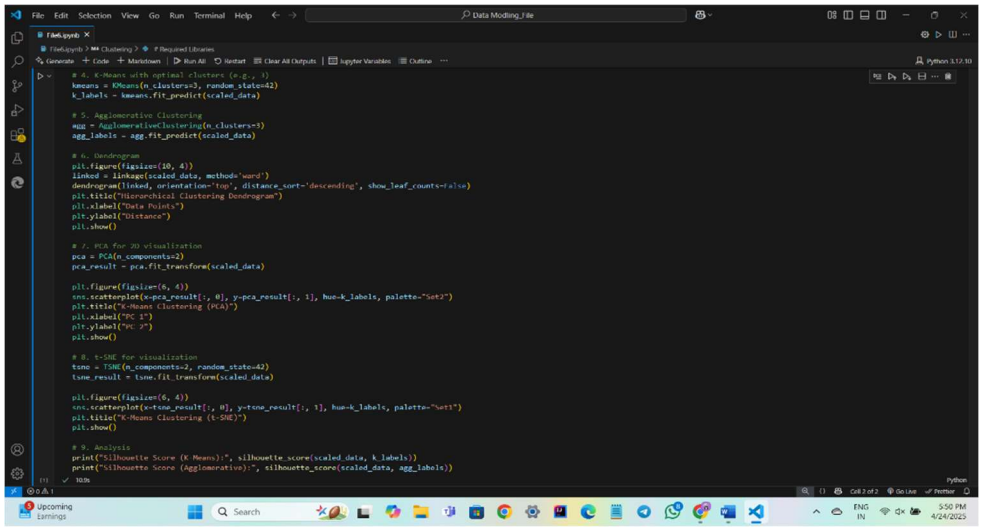
- ***Apply t-Distributed Stochastic Neighbor Embedding (t-SNE):***
    - Visualize the high-dimensional data in a 2D space.
    - Compare t-SNE vs. PCA results.

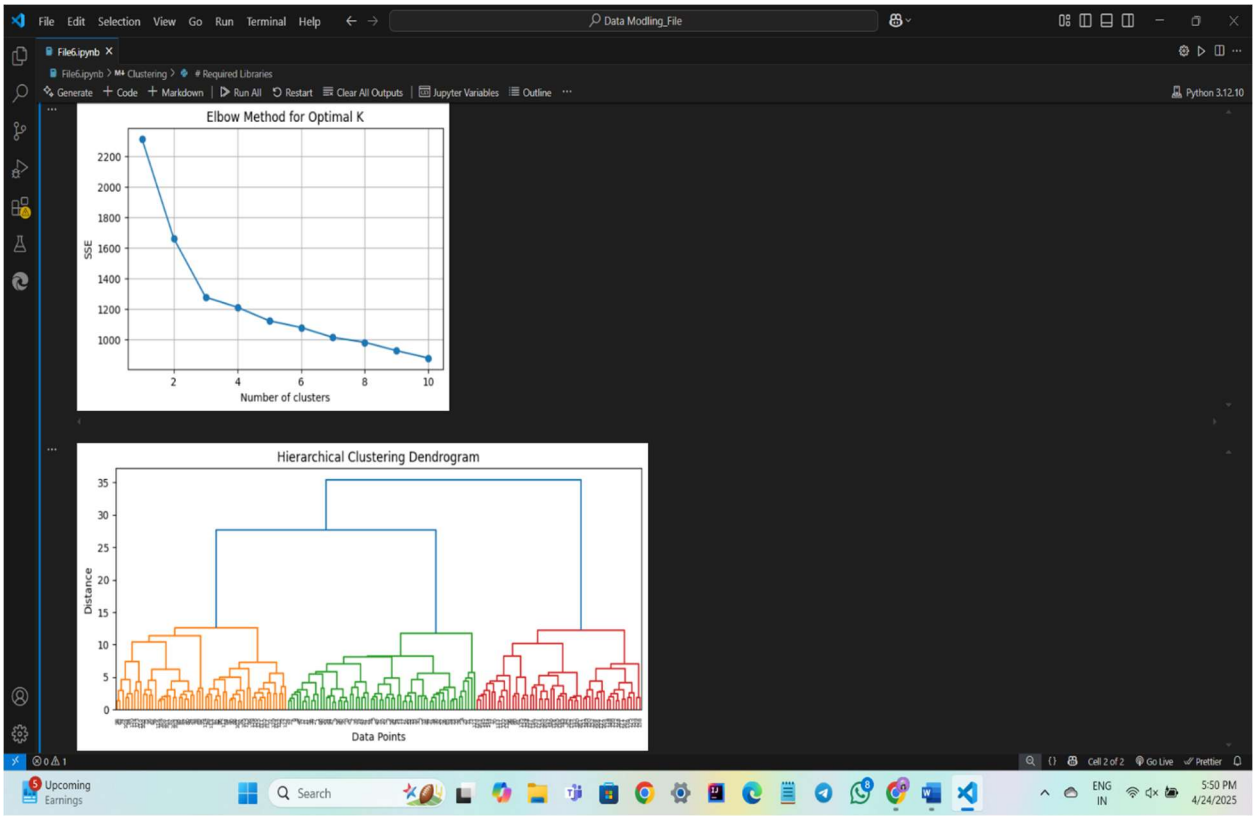
4. Analysis & Interpretation

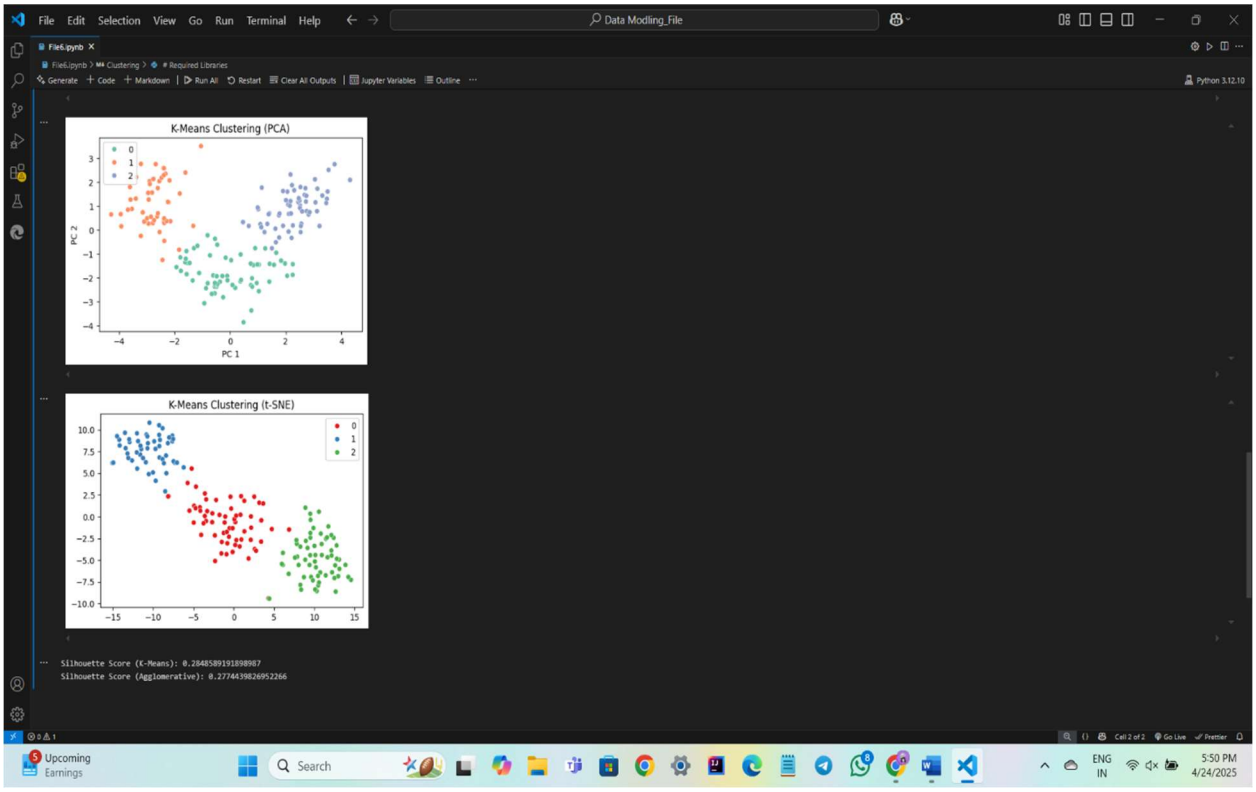
- Compare the performance of K-Means vs. Hierarchical Clustering.
- Discuss the effectiveness of PCA vs. t-SNE for visualization.
- Provide insights into clustering quality and visualization results**.**

Answer -:






File Edit Selection View Go Run Terminal Help
Data Modling_File
File6.ipynb
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.10
Elbow Method for Optimal K
SSE
Number of clusters
Hierarchical Clustering Dendrogram
Distance
Data Points
Upcoming Earnings
Search
ENG IN
5:50 PM 4/24/2025

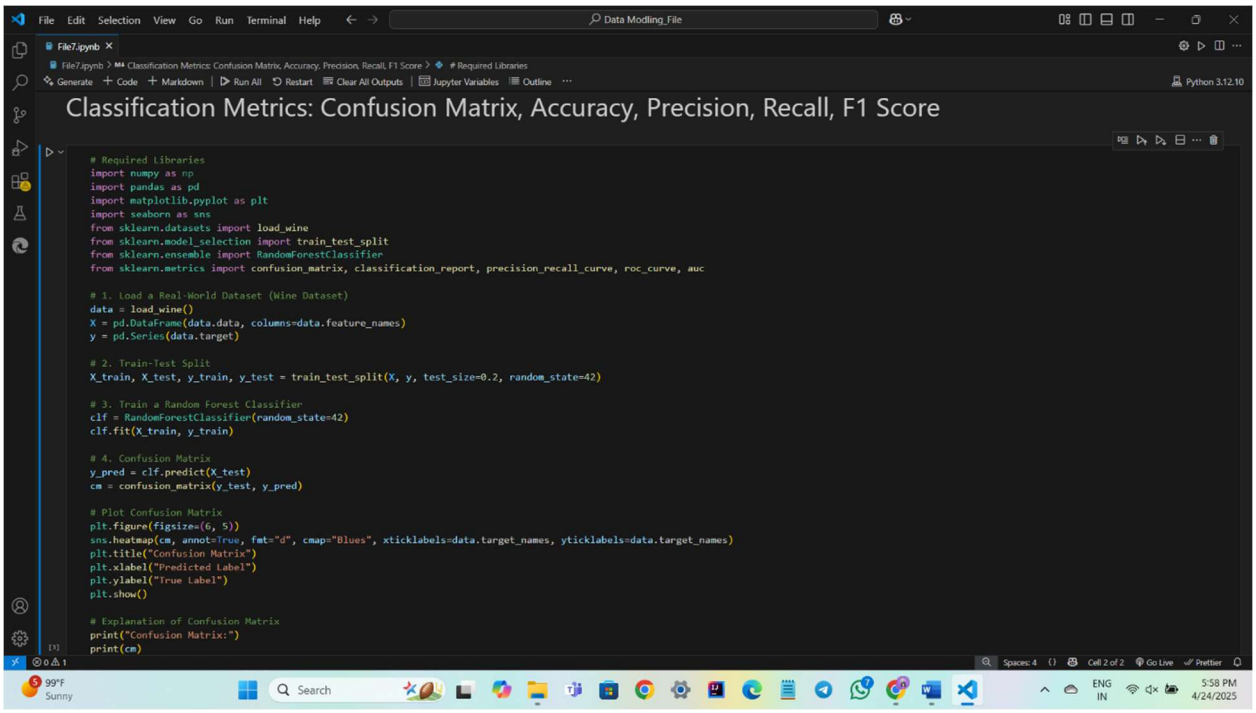

File Edit Selection View Go Run Terminal Help
Data Modling_File
File6.ipynb
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.10
K-Means Clustering (PCA)
PC 1
PC 2
K-Means Clustering (t-SNE)
Silhouette Score (K-Means): 0.2848589191898987
Silhouette Score (Agglomerative): 0.2774439826952266
Upcoming Earnings
Search
ENG IN
5:50 PM 4/24/2025

Experiment 8 -: Classification Metrics: Confusion Matrix, Accuracy, Precision, Recall, F1 Score

Experiment -:

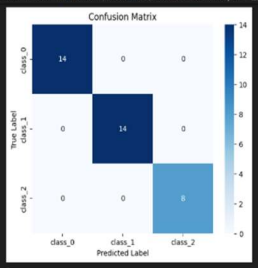
1. Train a classification model on a real-world dataset and compute the confusion matrix. What do the values in each cell represent?
2. Given a highly imbalanced dataset, why might accuracy not be a reliable metric? Compute precision, recall, and F1 score to support your argument.
3. How does changing the decision threshold of a classifier affect precision and recall? Demonstrate with an experiment.
4. Plot the ROC curve for a classification model and compute the AUC score. What does the AUC value tell you about the model's performance?
5. Compare the ROC curves of two different models. How do you determine which model is better? Can a model have high accuracy but a low AUC score? Explain with an example

Answer -:

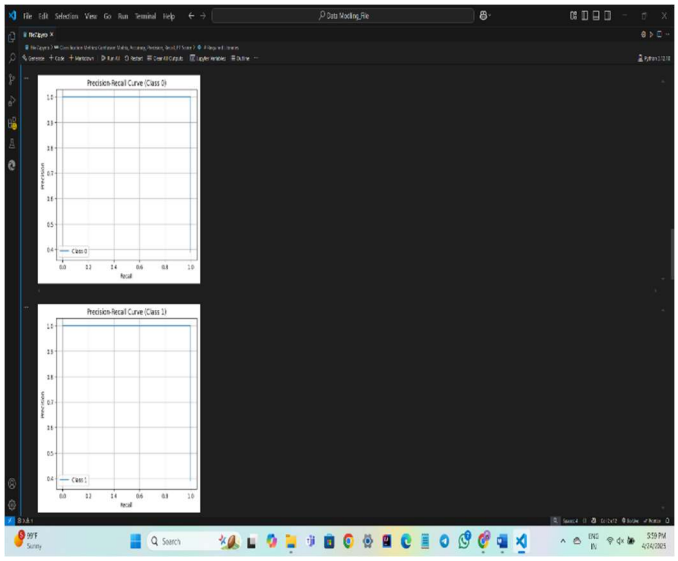


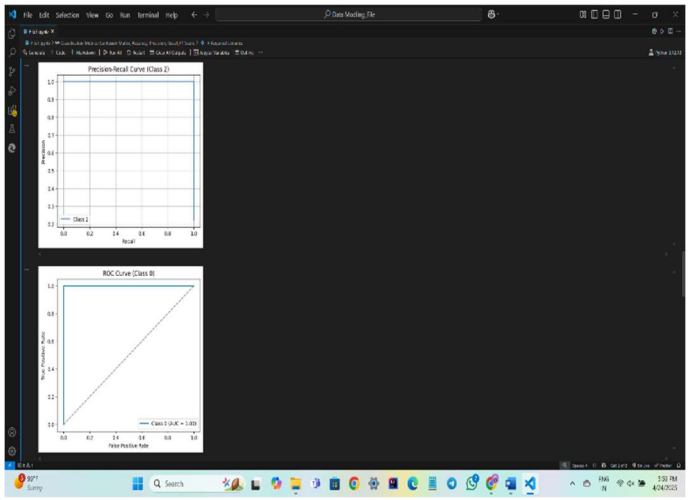
```python
# 5. Precision, Recall, and F1 Score (for imbalanced dataset)
print("\nClassification Report (Precision, Recall, F1):")
print(classification_report(y_test, y_pred, target_names=data.target_names))

from sklearn.preprocessing import label_binarize

# Binarize the output labels for multiclass classification
y_test_binarized = label_binarize(y_test, classes=[0, 1, 2])
n_classes = y_test_binarized.shape[1]

# 6. Changing the Decision Threshold (One-vs-Rest)
for i in range(n_classes):
    y_probs = clf.predict_proba(X_test)[:, i]
    precision, recall, thresholds = precision_recall_curve(y_test_binarized[:, i], y_probs)

    # Plot Precision-Recall Curve for each class
    plt.figure(figsize=(6, 5))
    plt.plot(recall, precision, label=f"Class {i}")
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'Precision-Recall Curve (Class {i})')
    plt.legend()
    plt.grid(True)
    plt.show()

# 7. ROC Curve and AUC (One-vs-Rest)
for i in range(n_classes):
    y_probs = clf.predict_proba(X_test)[:, i]
    fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_probs)
    roc_auc = auc(fpr, tpr)

    # Plot ROC Curve for each class
    plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, lw=2, label=f'Class {i} (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve (Class {i})')
    plt.legend(loc='lower right')
    plt.show()
```

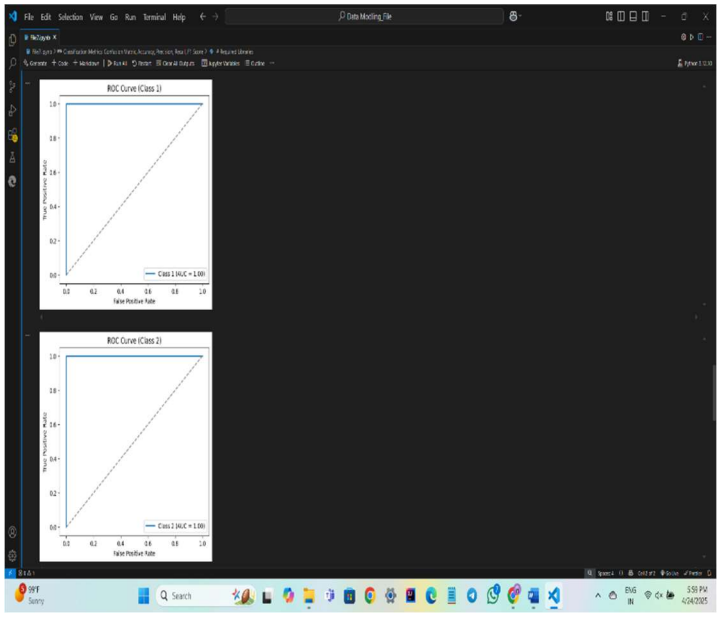
```python
# 8. Compare ROC Curves of Two Models
# Train a second model (e.g., Logistic Regression)
from sklearn.linear_model import LogisticRegression
lr_clf = LogisticRegression(max_iter=10000, random_state=42)
lr_clf.fit(X_train, y_train)

# Compute ROC curves and AUC for each class (One-vs-Rest)
for i in range(n_classes):
    # Random Forest
    rf_y_probs = clf.predict_proba(X_test)[:, i]
    fpr_rf, tpr_rf, _ = roc_curve(y_test_binarized[:, i], rf_y_probs)
    roc_auc_rf = auc(fpr_rf, tpr_rf)

    # Logistic Regression
    lr_y_probs = lr_clf.predict_proba(X_test)[:, i]
    fpr_lr, tpr_lr, _ = roc_curve(y_test_binarized[:, i], lr_y_probs)
    roc_auc_lr = auc(fpr_lr, tpr_lr)

    # Plot both ROC curves for the current class
    plt.figure(figsize=(6, 5))
    plt.plot(fpr_rf, tpr_rf, color='b', lw=2, label=f'Random Forest (AUC = {roc_auc_rf:.2f})')
    plt.plot(fpr_lr, tpr_lr, color='g', lw=2, label=f'Logistic Regression (AUC = {roc_auc_lr:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Comparison of ROC Curves (Class {i})')
    plt.legend(loc='lower right')
    plt.show()

# 9. Evaluation and Discussion
print("ROC AUC scores for each class:")
for i in range(n_classes):
    rf_y_probs = clf.predict_proba(X_test)[:, i]
    roc_auc_rf = auc(roc_curve(y_test_binarized[:, i], rf_y_probs)[0], roc_curve(y_test_binarized[:, i], rf_y_probs)[1])
    lr_y_probs = lr_clf.predict_proba(X_test)[:, i]
    roc_auc_lr = auc(roc_curve(y_test_binarized[:, i], lr_y_probs)[0], roc_curve(y_test_binarized[:, i], lr_y_probs)[1])
    print(f"Class {i} - Random Forest AUC: {roc_auc_rf:.2f}, Logistic Regression AUC: {roc_auc_lr:.2f}")
```

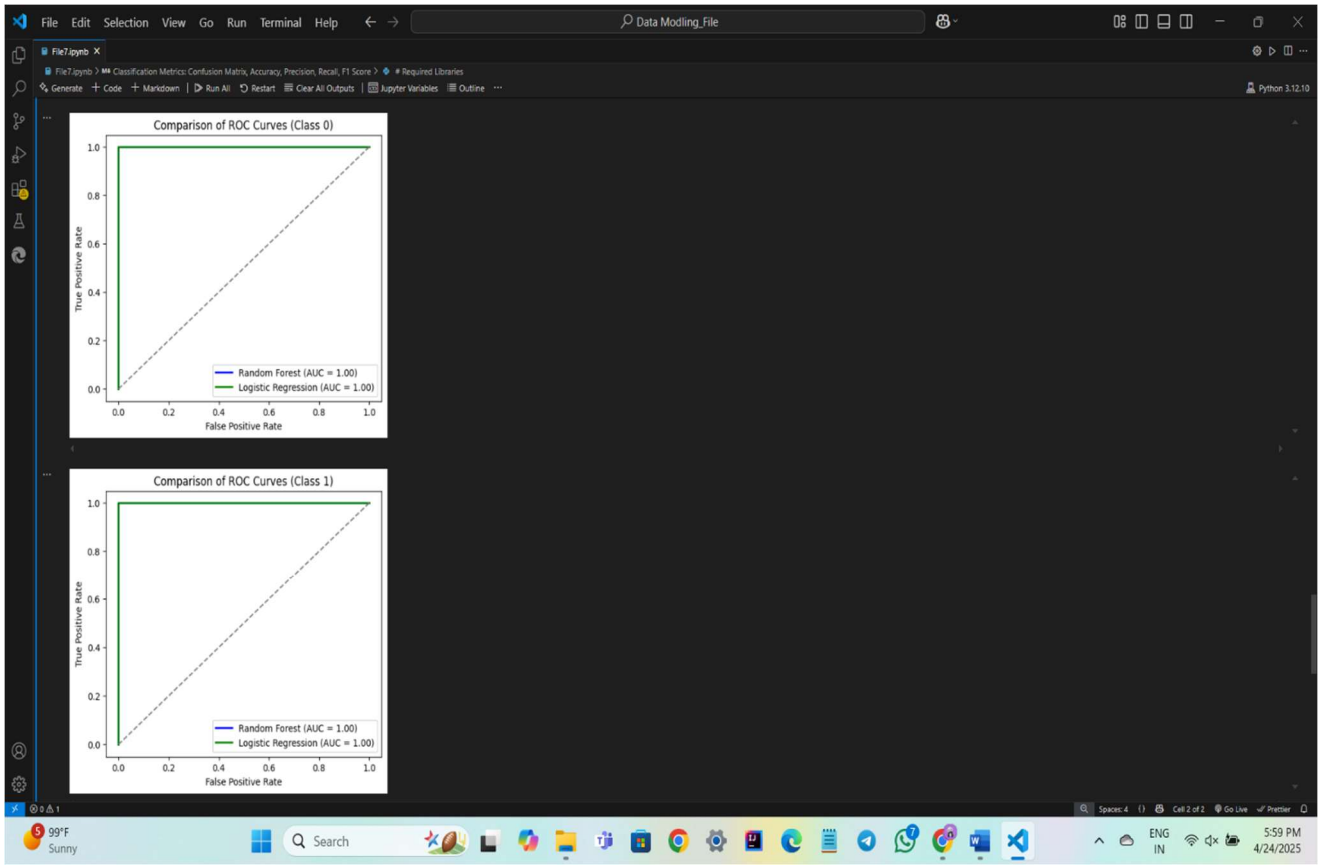



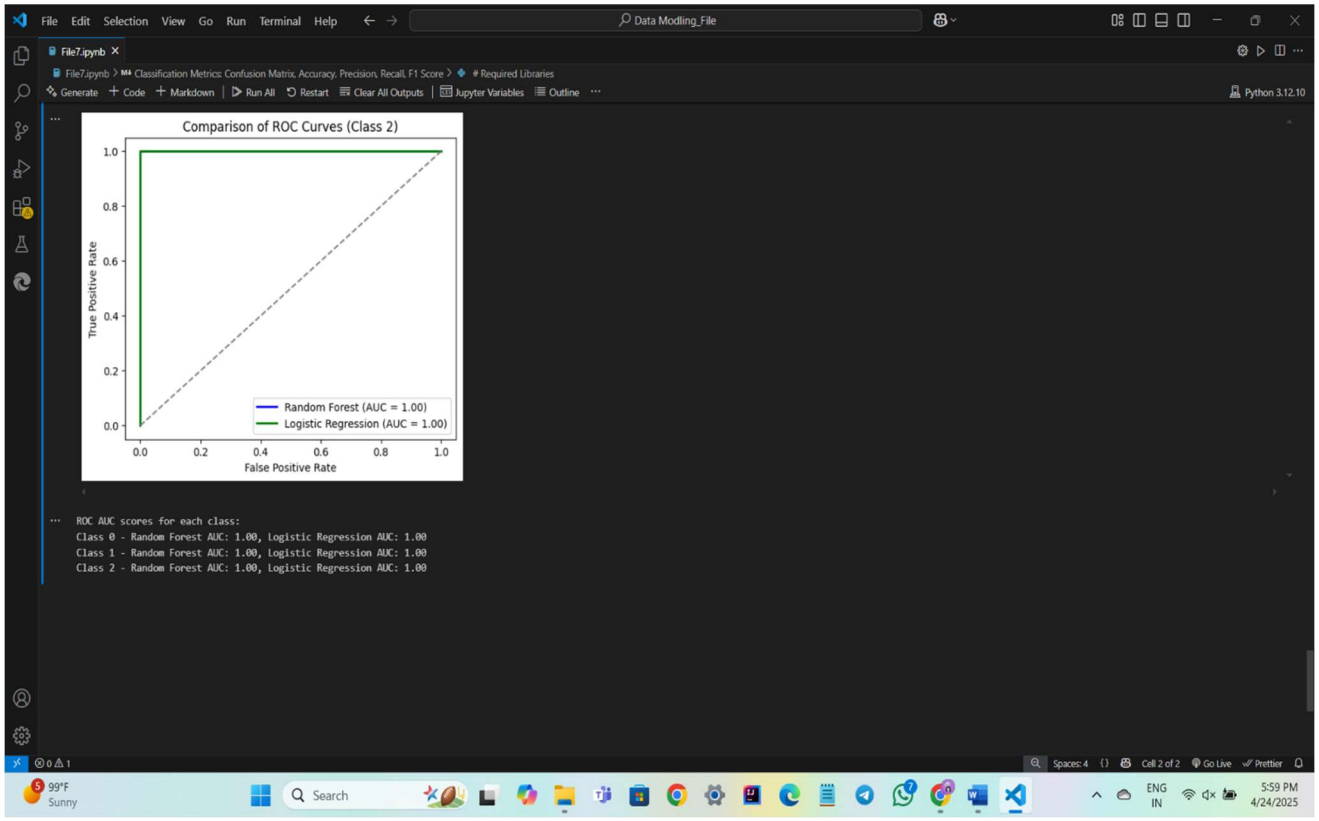

```
Confusion Matrix:
[[14  0  0]
 [ 0 14  0]
 [ 0  0  8]]

Classification Report (Precision, Recall, F1):
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        14
     class_1       1.00      1.00      1.00        14
     class_2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```


Precision-Recall Curve (Class 0)
Precision-Recall Curve (Class 1)


Precision-Recall Curve (Class 2)
ROC Curve (Class 0)


ROC Curve (Class 1)
ROC Curve (Class 2)


Comparison of ROC Curves (Class 0)
True Positive Rate
False Positive Rate
Random Forest (AUC = 1.00)
Logistic Regression (AUC = 1.00)
Comparison of ROC Curves (Class 1)
True Positive Rate
False Positive Rate
Random Forest (AUC = 1.00)
Logistic Regression (AUC = 1.00)


Comparison of ROC Curves (Class 2)
True Positive Rate
False Positive Rate
Random Forest (AUC = 1.00)
Logistic Regression (AUC = 1.00)
ROC AUC scores for each class:
Class 0 - Random Forest AUC: 1.00, Logistic Regression AUC: 1.00
Class 1 - Random Forest AUC: 1.00, Logistic Regression AUC: 1.00
Class 2 - Random Forest AUC: 1.00, Logistic Regression AUC: 1.00

Experiment 9 -: Time Series Modeling ARIMA, SARIMA, and Prophet

Experiment -:

- **Implementing SARIMA for Seasonal Data**

Steps:

1. Choose a dataset with seasonal patterns.
2. Perform seasonal decomposition to analyze trends and seasonality.
3. Fit a SARIMA model using seasonal_order parameters.
4. Forecast future values and compare with actual data

- **Prophet for Time Series Forecasting**

Steps:

1. Install and import Facebook's Prophet library.
2. Prepare data in Prophet format (ds and y columns).
3. Fit a Prophet model and forecast future data points.
4. Plot forecast trends and confidence intervals.

- **Hyperparameter Tuning Using Grid Search**

Step:

1. Define a machine learning model (e.g., RandomForest, XGBoost).
2. Use GridSearchCV to find the best hyperparameters.
3. Evaluate model performance using cross-validation.

Answer -:

File Edit Selection View Go Run Terminal Help — Data Modling_File

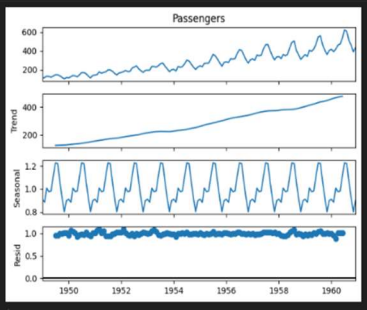
File8.ipynb

Time Series Modeling ARIMA, SARIMA, and Prophet

1.Implementing SARIMA for Seasonal Data

```
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Load the AirPassengers dataset from a CSV file
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv"
data = pd.read_csv(url)
data['Month'] = pd.to_datetime(data['Month'])
data.set_index('Month', inplace=True)

# Decompose the time series
decomposition = sm.tsa.seasonal_decompose(data['Passengers'], model='multiplicative')
decomposition.plot()
plt.show()
```

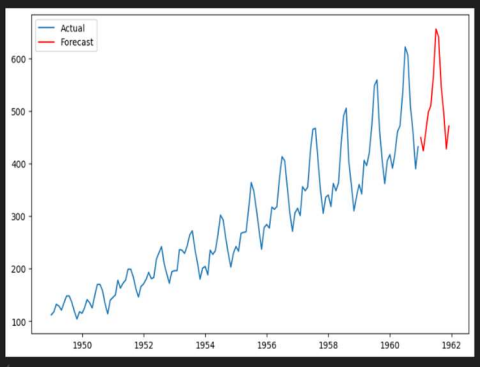
[4] ✓ 2.4s Python




```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Fit SARIMA model
model = SARIMAX(data['Passengers'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
results = model.fit()

# Forecast future values
forecast = results.get_forecast(steps=12)
forecast_values = forecast.predicted_mean

# Plot forecast vs actual values
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['Passengers'], label='Actual')
plt.plot(forecast_values.index, forecast_values, label='Forecast', color='red')
plt.legend()
plt.show()
```

[9] ✓ 0.7s Python

```
C:\Users\hp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\hp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
```



97°F Sunny — Search — ENG IN — 6:25 PM 4/24/2025

File Edit Selection View Go Run Terminal Help

Data Modling_File

File8.ipynb

File8.ipynb > Time Series Modeling ARIMA, SARIMA, and Prophet > 3. Hyperparameter Tuning Using Grid Search > from sklearn.model_selection import GridSearchCV

Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ...

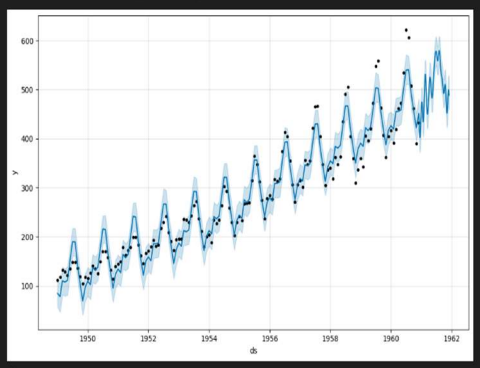
Python 3.12.10

2. Prophet for Time Series Forecasting

```
from prophet import Prophet

# Prepare data for Prophet
prophet_data = data.reset_index()  # Reset index to use Month as a column
prophet_data.rename(columns={'Month': 'ds', 'Passengers': 'y'}, inplace=True)

# Fit the Prophet model
model = Prophet()
model.fit(prophet_data)

# Create future dataframe (forecast 365 days ahead)
future = model.make_future_dataframe(periods=365)
forecast = model.predict(future)

# Plot forecast with confidence intervals
model.plot(forecast)
plt.show()
```

[32] ✓ 3.7s Python

```
18:23:43 - cmdstanpy - INFO - Chain [1] start processing
18:23:44 - cmdstanpy - INFO - Chain [1] done processing
```



0 4 Cell 9 of 9 Go Live

97°F Sunny Search ENG IN 6:25 PM 4/24/2025

File Edit Selection View Go Run Terminal Help

Data Modling_File

File8.ipynb

File8.ipynb > Time Series Modeling ARIMA, SARIMA, and Prophet > 3. Hyperparameter Tuning Using Grid Search > from sklearn.model_selection import GridSearchCV

Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ...

Python 3.12.10

3. Hyperparameter Tuning Using Grid Search

```
from sklearn.ensemble import RandomForestRegressor

# Define model
rf_model = RandomForestRegressor()
```

[33] ✓ 0.8s Python

```
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Apply GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3)
from sklearn.model_selection import train_test_split

# Prepare features and target variable
X = data.index.factorize()[0].reshape(-1, 1)  # Convert datetime index to numerical values
y = data['Passengers']

# 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)

# Fit the model
grid_search.fit(X_train, y_train)

# Print the best parameters
print("Best Parameters:", grid_search.best_params_)
```

[35] ✓ 11.9s Python

```
Best Parameters: {'max_depth': 30, 'min_samples_split': 2, 'n_estimators': 100}
```

0 4 Cell 9 of 9 Go Live Prettier

97°F Sunny Search ENG IN 6:26 PM 4/24/2025

Experiment 10: Model Validation Techniques

Experiment -:

1. **Implementing -Stratified K-Fold Cross-Validation**

Steps:

a) Load an imbalanced classification dataset (e.g., breast cancer dataset).

b) Apply Stratified K-Fold to ensure equal class distribution in each fold.

c) Compare results with standard K-Fold

2. **Implementing -Time-Based Split for Time Series Data**

Step:

a) Load a time series dataset (e.g., stock prices, weather data)

b) Perform a time-based split where training data is from past timestamps, and testing data from future timestamps.

c) Compare results with a random split.

3. **Ensuring Reproducibility in Machine Learning Experiments.**

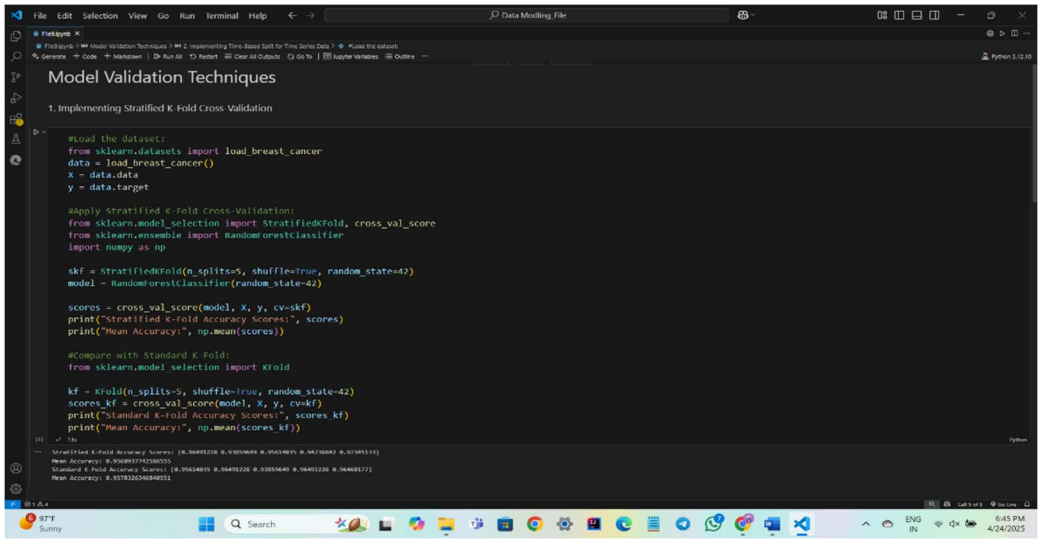
Step:

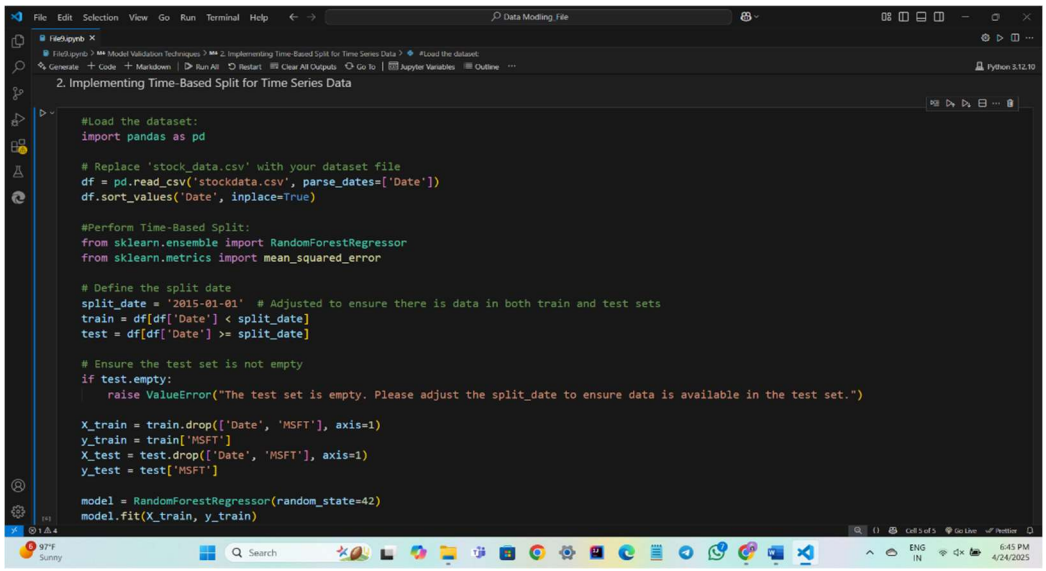
a) Set random seeds for NumPy, TensorFlow, and Scikit-learn.

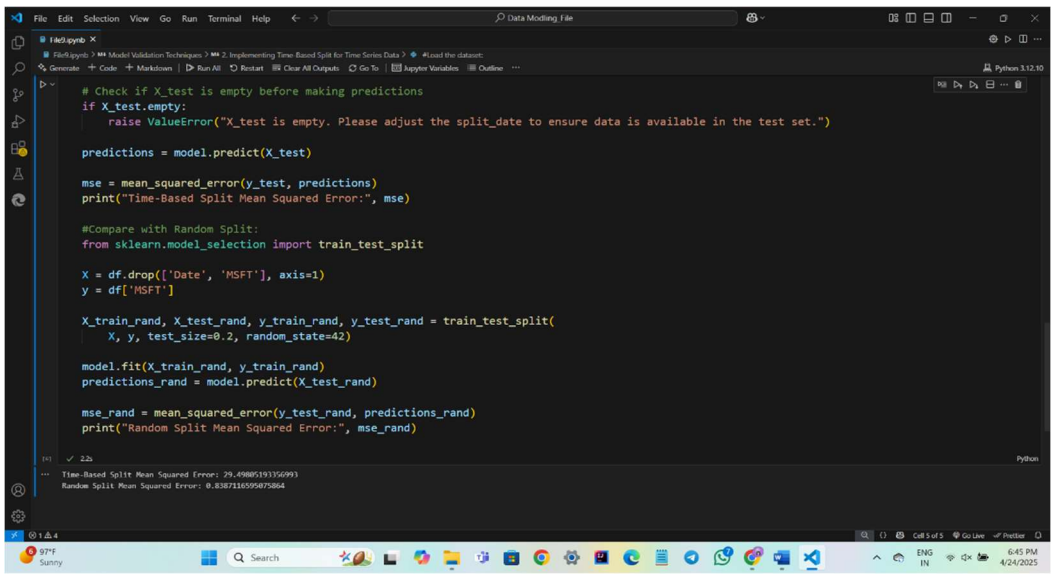
b) Use DVC (Data Version Control) for dataset tracking.

c) Store model training scripts in Git for version control.

Answer -:


Model Validation Techniques
1. Implementing Stratified K-Fold Cross-Validation
#Load the dataset:
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X = data.data
y = data.target
#Apply Stratified K-Fold Cross-Validation:
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
import numpy as np
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
model = RandomForestClassifier(random_state=42)
scores = cross_val_score(model, X, y, cv=skf)
print("Stratified K-Fold Accuracy Scores:", scores)
print("Mean Accuracy:", np.mean(scores))
#Compare with Standard K Fold:
from sklearn.model_selection import KFold
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores_kf = cross_val_score(model, X, y, cv=kf)
print("Standard K-Fold Accuracy Scores:", scores_kf)
print("Mean Accuracy:", np.mean(scores_kf))
Stratified K-Fold Accuracy Scores: [0.96491228 0.93859649 0.95614035 0.94736842 0.97345133]
Mean Accuracy: 0.9560937742586555
Standard K-Fold Accuracy Scores: [0.95614035 0.96491228 0.93859649 0.96491228 0.96460177]
Mean Accuracy: 0.9578263468848551


2. Implementing Time-Based Split for Time Series Data
#Load the dataset:
import pandas as pd
# Replace 'stock_data.csv' with your dataset file
df = pd.read_csv('stockdata.csv', parse_dates=['Date'])
df.sort_values('Date', inplace=True)
#Perform Time-Based Split:
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
# Define the split date
split_date = '2015-01-01' # Adjusted to ensure there is data in both train and test sets
train = df[df['Date'] < split_date]
test = df[df['Date'] >= split_date]
# Ensure the test set is not empty
if test.empty:
raise ValueError("The test set is empty. Please adjust the split_date to ensure data is available in the test set.")
X_train = train.drop(['Date', 'MSFT'], axis=1)
y_train = train['MSFT']
X_test = test.drop(['Date', 'MSFT'], axis=1)
y_test = test['MSFT']
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)


# Check if X_test is empty before making predictions
if X_test.empty:
raise ValueError("X_test is empty. Please adjust the split_date to ensure data is available in the test set.")
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("Time-Based Split Mean Squared Error:", mse)
#Compare with Random Split:
from sklearn.model_selection import train_test_split
X = df.drop(['Date', 'MSFT'], axis=1)
y = df['MSFT']
X_train_rand, X_test_rand, y_train_rand, y_test_rand = train_test_split(
X, y, test_size=0.2, random_state=42)
model.fit(X_train_rand, y_train_rand)
predictions_rand = model.predict(X_test_rand)
mse_rand = mean_squared_error(y_test_rand, predictions_rand)
print("Random Split Mean Squared Error:", mse_rand)
Time-Based Split Mean Squared Error: 29.498051931356993
Random Split Mean Squared Error: 0.8387116595075864

Experiment 11 -: Overfitting and Underfitting

Experiment -:

1.Compare and Interpret Results:

- Evaluate how L1, L2, and Dropout affect model performance.
- Discuss which regularization technique works best and why

2.Setting Up Random Seeds:

- Implement a script where all necessary libraries (NumPy, TensorFlow, Scikit-Learn, etc.) have **fixed random seeds**.
- Run the same model multiple times and verify that the results are consistent

3. Version Control for Machine Learning Projects:

- Use **Git** to track changes in your model.
- Create different branches to experiment with different regularization techniques.
- Document model performance results for comparison.

4. **Using Reproducibility Tools:**

- Utilize tools like **MLflow** or **DVC (Data Version Control)** to track experiments.
- Save hyperparameters and evaluation metrics for future reference.
- Demonstrate how to reproduce previous experiments exactly.

Answer -:

1. Compare and Interpret Results: L1, L2, and Dropout Regularization

L1 Regularization (Lasso):

- Purpose: Encourages sparsity by driving some weights to zero, effectively performing feature selection.

- Use Case: Beneficial when you suspect that only a subset of features are important.
- Consideration: Can lead to sparse models, which are easier to interpret but might underfit if too aggressive.

L2 Regularization (Ridge):

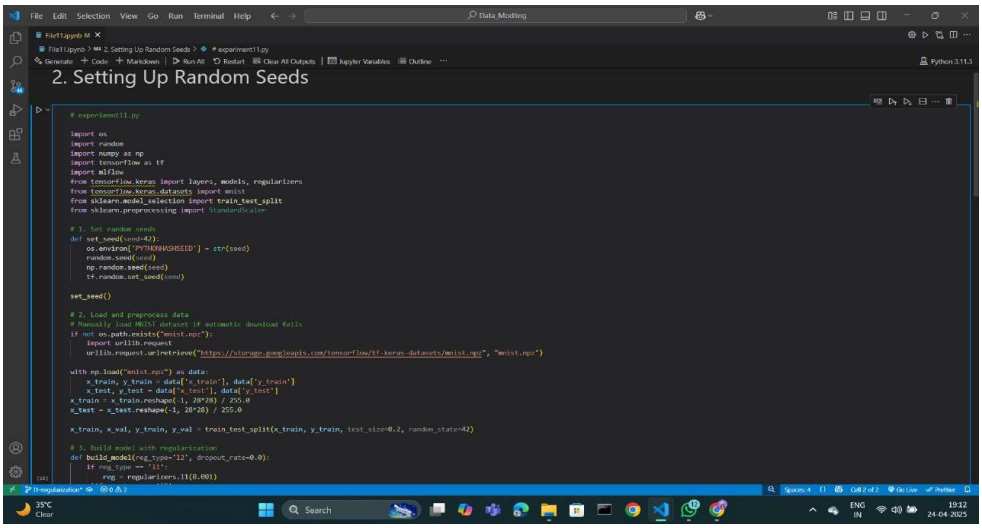
- Purpose: Penalizes large weights by adding the squared magnitude of coefficients as a penalty term to the loss function.
- Use Case: Helps when all features contribute to the output, but you want to prevent overfitting.
- Consideration: Tends to distribute the error among all weights, leading to smaller, more evenly distributed weights.

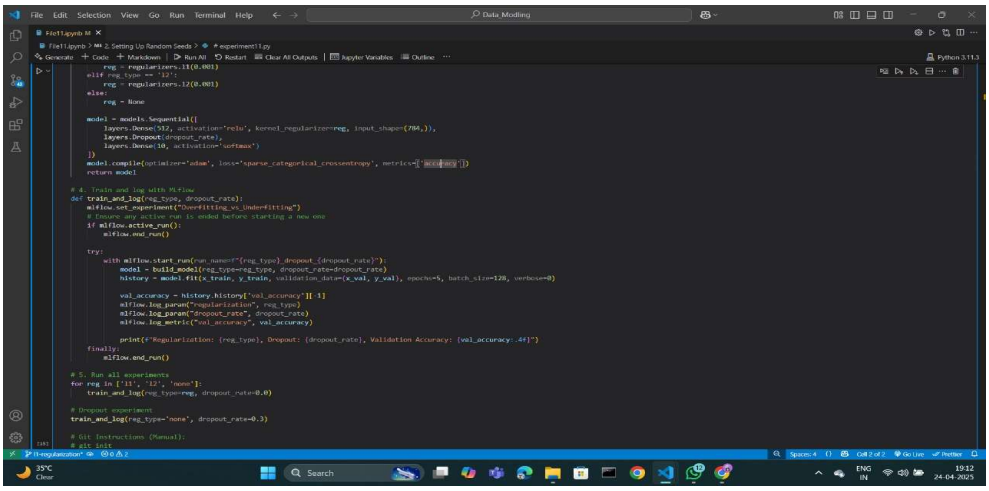
Dropout:

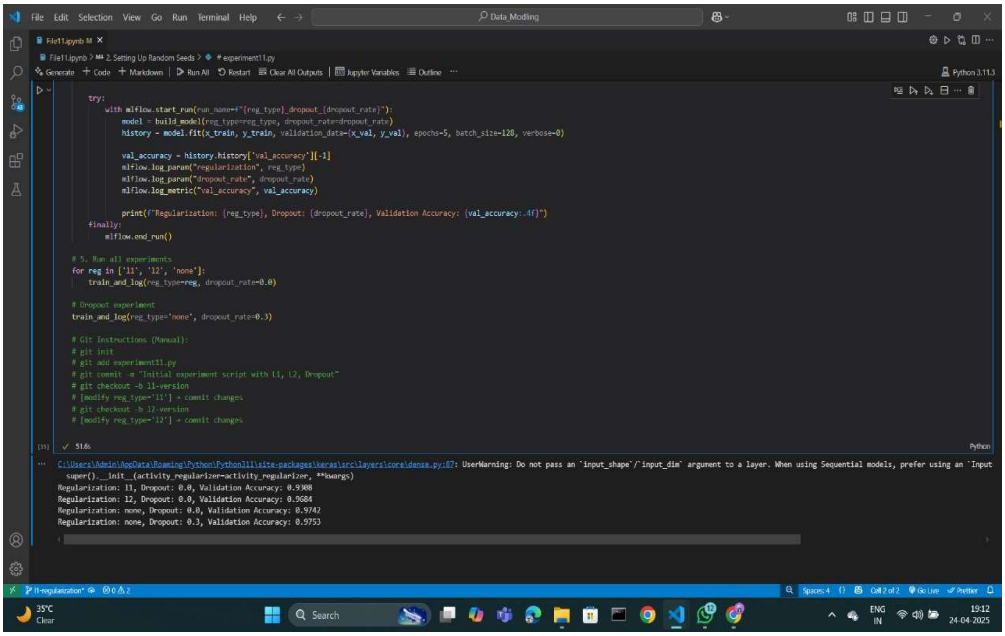
- Purpose: Randomly sets a fraction of input units to zero during training, which helps prevent overfitting by making the network less sensitive to specific weights.
- Use Case: Particularly effective in deep neural networks to prevent co-adaptation of neurons.
- Consideration: Not typically used in conjunction with L1 or L2 regularization; choosing between them depends on the specific problem and model architecture.

Comparison:

- L1 is useful for feature selection, L2 for weight decay, and Dropout for preventing co-adaptation in neural networks.
- The effectiveness of each regularization technique depends on the dataset and model complexity.


2. Setting Up Random Seeds
# experiment11.py
import os
import random
import numpy as np
import tensorflow as tf
import mlflow
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# 1. Set random seeds
def set_seed(seed=42):
os.environ['PYTHONHASHSEED'] = str(seed)
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
set_seed()
# 2. Load and preprocess data
# Manually load MNIST dataset if automatic download fails
if not os.path.exists("mnist.npz"):
import urllib.request
urllib.request.urlretrieve("https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz", "mnist.npz")
with np.load("mnist.npz") as data:
x_train, y_train = data['x_train'], data['y_train']
x_test, y_test = data['x_test'], data['y_test']
x_train = x_train.reshape(-1, 28*28) / 255.0
x_test = x_test.reshape(-1, 28*28) / 255.0
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)
# 3. Build model with regularization
def build_model(reg_type='l2', dropout_rate=0.0):
if reg_type == 'l1':
reg = regularizers.l1(0.001)


reg = regularizers.l1(0.001)
elif reg_type == 'l2':
reg = regularizers.l2(0.001)
else:
reg = None
model = models.Sequential([
layers.Dense(512, activation='relu', kernel_regularizer=reg, input_shape=(784,)),
layers.Dropout(dropout_rate),
layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
return model
# 4. Train and log with MLflow
def train_and_log(reg_type, dropout_rate):
mlflow.set_experiment("Overfitting_vs_Underfitting")
# Ensure any active run is ended before starting a new one
if mlflow.active_run():
mlflow.end_run()
try:
with mlflow.start_run(run_name=f"{reg_type}_dropout_{dropout_rate}"):
model = build_model(reg_type=reg_type, dropout_rate=dropout_rate)
history = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=5, batch_size=128, verbose=0)
val_accuracy = history.history['val_accuracy'][-1]
mlflow.log_param("regularization", reg_type)
mlflow.log_param("dropout_rate", dropout_rate)
mlflow.log_metric("val_accuracy", val_accuracy)
print(f"Regularization: {reg_type}, Dropout: {dropout_rate}, Validation Accuracy: {val_accuracy:.4f}")
finally:
mlflow.end_run()
# 5. Run all experiments
for reg in ['l1', 'l2', 'none']:
train_and_log(reg_type=reg, dropout_rate=0.0)
# Dropout experiment
train_and_log(reg_type='none', dropout_rate=0.3)
# Git Instructions (Manual):
# git init


try:
with mlflow.start_run(run_name=f"{reg_type}_dropout_{dropout_rate}"):
model = build_model(reg_type=reg_type, dropout_rate=dropout_rate)
history = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=5, batch_size=128, verbose=0)
val_accuracy = history.history['val_accuracy'][-1]
mlflow.log_param("regularization", reg_type)
mlflow.log_param("dropout_rate", dropout_rate)
mlflow.log_metric("val_accuracy", val_accuracy)
print(f"Regularization: {reg_type}, Dropout: {dropout_rate}, Validation Accuracy: {val_accuracy:.4f}")
finally:
mlflow.end_run()
# 5. Run all experiments
for reg in ['l1', 'l2', 'none']:
train_and_log(reg_type=reg, dropout_rate=0.0)
# Dropout experiment
train_and_log(reg_type='none', dropout_rate=0.3)
# Git Instructions (Manual):
# git init
# git add experiment11.py
# git commit -m "Initial experiment script with l1, l2, Dropout"
# git checkout -b l1-version
# [modify reg_type='l1'] → commit changes
# git checkout -b l2-version
# [modify reg_type='l2'] → commit changes
✓ 51.6s
Python
C:\Users\Admin\AppData\Roaming\Python\Python311\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Regularization: l1, Dropout: 0.0, Validation Accuracy: 0.9708
Regularization: l2, Dropout: 0.0, Validation Accuracy: 0.9684
Regularization: none, Dropout: 0.0, Validation Accuracy: 0.9742
Regularization: none, Dropout: 0.3, Validation Accuracy: 0.9753