DEPARTMENT OF COMPUTER SCIENCE ENGINEERING -DATA SCIENCE

PREDICTIVE ANALYTICS LAB MANUAL

NAME OF THE LABORATORY	**:**	**PREDICTIVE ANALYTICS LAB**
YEAR AND SEM	:	**IV B.TECH I SEM**
REGULATION/LAB CODE	:	**KR21/KG21CD705**

COMPUTER SCIENCE AND ENGINEERING – DATA SCIENCE

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

VISION AND MISSION OF THE INSTITUTION

VISION:

To become self-sustainable institution which is recognized for its new age engineering through innovative teaching and learning culture, inculcating research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION:

- To offer undergraduate and post-graduate programs that is supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
- To provide necessary support structures for students, which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
- To provide faculty and students with an ecosystem that fosters research and development through strategic partnerships with government organisations and collaboration with industries.
- To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic growth.

VISION AND MISSION OF DEPARTMENT

VISION

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION

- To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
- To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
- To encourage professional development of students that will inculcate ethical values and leadership skills while working with the community to address societal issues.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING- DATA SCIENCE

Program Educational Objectives (PEOs):

A graduate of the Computer Science and Engineering Program should:

Program Educational Objective 1: (PEO1)

The Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.

Program Educational Objective 2: (PEO2)

***The** Graduates have successful careers in computer science and engineering fields or will be able to Successfully pursue advanced degrees.*

Program Educational Objective 3: (PEO3)

The Graduates will communicate effectively, work collaboratively and exhibit high levels of Professionalism, moral and ethical responsibility.

Program Educational Objective 4: (PEO4)

***The** Graduates will develop the ability to understand and analyse Engineering issues in a broader perspective with ethical responsibility towards sustainable development.*

Program Outcomes (POs):

PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs):

PSO1	**Problem Solving Skills** – Graduate will be able to apply computational techniques and software principles to solve complex engineering problems pertaining to software engineering.
PSO2	**Professional Skills** – Graduate will be able to think critically, communicate effectively, and collaborate in teams through participation in co and extra-curricular activities.
PSO3	**Successful Career** – Graduates will possess a solid foundation in computer science and engineering that will enable them to grow in their profession and pursue lifelong learning through post-graduation and professional development.

Week 1: Load Datasets and Perform Data Pre-Processing

Aim:

- Load various datasets for exploration and analysis.
- Implement pre-processing techniques to handle missing values and outliers.
- Apply data normalization to enhance feature comparability.

Program:

```
import pandas as pd
from sklearn.impute import SimpleImputer  # Handle missing values
from sklearn.preprocessing import StandardScaler, MinMaxScaler  # Normalize data

# Load a sample dataset
data = pd.read_csv("your_dataset.csv")

# Handle missing values (assuming numerical columns)
imputer = SimpleImputer(strategy="mean")
data_imputed = pd.DataFrame(imputer.fit_transform(data))

# Identify and handle outliers (e.g., using IQR or Z-score)
# ... (Replace with your outlier detection and handling logic)

# Choose a normalization method (e.g., standard or min-max)
scaler = StandardScaler()  # Consider MinMaxScaler for bounded features
data_normalized = pd.DataFrame(scaler.fit_transform(data_imputed))

# Output pre-processed data
print("Pre-processed data:")
print(data_normalized.head())
```

Week 2: Feature Extraction using PCA and LDA Techniques

Aim:

- Reduce dimensionality using Principal Component Analysis (PCA).
- Extract discriminative features using Linear Discriminant Analysis (LDA).

Program:

```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Extract components using PCA (adjust n_components as needed)
pca = PCA(n_components=2)
features_pca = pca.fit_transform(data_normalized)  # Use pre-processed data

# Extract discriminative features using LDA (assuming labeled data)
lda = LinearDiscriminantAnalysis()
features_lda = lda.fit_transform(data_normalized, data["target_label"])  # Replace with labels

# Analyze and interpret extracted features
# ... (Explore component loadings, visualize feature distributions, etc.)
```

Output:

```
Reduced-dimensionality data using PCA:
```

(Sample of data with reduced dimensions)

```
Discriminative features using LDA:
```

(Sample of data with features optimized for class separation)

Week 3: Data Analysis I (Train-Test Split, Feature Selection) and Working with Scikit-Learn

Aim:

- Split data into training and testing sets for model evaluation.
- Explore various feature selection techniques and assess their impact.

Program:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2, f_classif  # Example techniques

# Split data into training and testing sets (adjust test_size as needed)
X_train, X_test, y_train, y_test = train_test_split(features_pca, data["target_label"], test_size=0.2)

# Apply feature selection (replace with your chosen technique and evaluation)
selector = SelectKBest(score_func=chi2, k=5)  # K best features based on chi-squared
features_selected = selector.fit_transform(X_train, y_train)

# Evaluate feature selection impact using model performance (e.g., accuracy)
# ... (Train models on both original and selected features, compare results)
```

Output:

```
Training and testing data shapes:
... (Show shapes of X_train, X_test, y_train, y_test)

Selected features:
```

(Sample of selected features)

Week 4: Decision Tree Classifier using Information Gain and Entropy with Tree Pruning

Aim:

- Implement a decision tree classifier using Information Gain and Entropy metrics for feature selection.
- Evaluate model performance and apply tree pruning to improve generalization.

Program:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Load and pre-process your data (using similar logic from Week 1)
# ...

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y_target,
test_size=0.2)

# Create decision trees with different criteria (Information Gain and Entropy)
dt_gini = DecisionTreeClassifier(criterion="gini")
dt_entropy = DecisionTreeClassifier(criterion="entropy")

# Train and evaluate models on the training set
dt_gini.fit(X_train, y_train)
dt_entropy.fit(X_train, y_train)

# Train accuracy
train_accuracy_gini = accuracy_score(y_train, dt_gini.predict(X_train))
train_accuracy_entropy = accuracy_score(y_train, dt_entropy.predict(X_train))

# Test accuracy
test_accuracy_gini = accuracy_score(y_test, dt_gini.predict(X_test))
test_accuracy_entropy = accuracy_score(y_test, dt_entropy.predict(X_test))

# Print performance metrics
```

```
print("Decision Tree (Gini): Train accuracy:", train_accuracy_gini, "Test accuracy:", test_accuracy_gini)
print("Decision Tree (Entropy): Train accuracy:", train_accuracy_entropy, "Test accuracy:", test_accuracy_entropy)

# Implement tree pruning using `DecisionTreeClassifier` options or custom pruning logic
# ... (Example using max_depth parameter)
pruned_tree = DecisionTreeClassifier(criterion="gini", max_depth=3)
pruned_tree.fit(X_train, y_train)
pruned_test_accuracy = accuracy_score(y_test, pruned_tree.predict(X_test))

# Evaluate pruned tree performance
print("Pruned Decision Tree: Test accuracy:", pruned_test_accuracy)

# Visualize decision trees using `plot_tree` or other libraries
# ...
```

Week 5: Work with Similarity and Dissimilarity Metrics and Apply K-Nearest Neighbors algorithm.

Aim:

- Implement the KNN algorithm for both classification and regression tasks.
- Experiment with different distance metrics and `k` values to optimize performance.

Program:

```
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.model_selection import train_test_split

# Load and pre-process your data (similar to Week 1)
# ...

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y_target,
test_size=0.2)

# Classification

# Create KNN classifier with different distance metrics and k values
knn_clf_euclidean = KNeighborsClassifier(n_neighbors=5, metric="euclidean")
knn_clf_manhattan = KNeighborsClassifier(n_neighbors=3, metric="manhattan")
knn_clf_minkowski = KNeighborsClassifier(n_neighbors=7, metric="minkowski", p=2)

# Train and evaluate models
knn_clf_euclidean.fit(X_train, y_train)
knn_clf_manhattan.fit(X_train, y_train)
knn_clf_minkowski.fit(X_train, y_train)

# Test accuracy
test_accuracy_euclidean = accuracy_score(y_test,
knn_clf_euclidean.predict(X_test))
```

```
test_accuracy_manhattan = accuracy_score(y_test,
knn_clf_manhattan.predict(X_test))
test_accuracy_minkowski = accuracy_score(y_test,
knn_clf_minkowski.predict(X_test))

# Print performance metrics
print("KNN (Euclidean): Test accuracy:", test_accuracy_euclidean)
print("KNN (Manhattan): Test accuracy:", test_accuracy_manhattan)
print("KNN (Minkowski p=2): Test accuracy:", test_accuracy_minkowski)

# Regression

# Create KNN regressor with different distance metrics and k values
knn_reg_euclidean = KNeighborsRegressor(n_neighbors=5, metric="euclidean")
knn_reg_manhattan = KNeighborsRegressor(n_neighbors=3, metric="manhattan")
knn_reg_minkowski = KNeighborsRegressor(n_neighbors=7, metric="minkowski",
p=2)

# Train and evaluate models
knn_reg_euclidean.fit(X_train, y_train)
knn_reg_manhattan.fit(X_train, y_train)
knn_reg_minkowski.fit(X_train, y_train)

# Test mean squared error (MSE)
test_mse_euclidean = mean_squared_error(y_test,
knn_reg_euclidean.predict(X_test))
test_mse_manhattan = mean_squared_error(y_test,
knn_reg_manhattan.predict(X_test))
test_mse_minkowski = mean_squared_error(y_test,
knn_reg_minkowski.predict(X_test))

# Print performance metrics
print("KNN (Euclidean): Test MSE:", test_mse_euclidean)
print("KNN (Manhattan): Test MSE:", test_mse_manhattan)
print("KNN (Minkowski p=2): Test MSE:", test_mse_minkowski)

# ... (Optional: Visualize predictions using techniques like k-NN distance plots)
```

Week 6: Similarity and Dissimilarity Metrics with Output

Aim:

- Explore various similarity and dissimilarity metrics like Euclidean distance, cosine similarity, and Jaccard similarity to calculate pairwise distances between data points.
- Apply these metrics to tasks like clustering or nearest neighbor search.

Program:

```
from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import euclidean, cosine, jaccard
import pandas as pd

# Load and pre-process your data (similar to Week 1)
# ...

# Calculate pairwise distances using various metrics
euclidean_dists = pairwise_distances(X_preprocessed, metric="euclidean")
cosine_dists = pairwise_distances(X_preprocessed, metric="cosine")
jaccard_dists = pairwise_distances(X_preprocessed, metric="jaccard")

# Output distance matrices and insights

# Euclidean distance matrix (sample)
print("Sample of Euclidean distance matrix:")
print(pd.DataFrame(euclidean_dists[:5, :5], columns=X_preprocessed.columns))

# Analyze Euclidean distances
# ... (Calculate minimum, maximum, mean distances, visualize using heatmaps)

# Cosine similarity matrix (sample)
print("Sample of Cosine similarity matrix:")
print(pd.DataFrame(cosine_dists[:5, :5], columns=X_preprocessed.columns))
```

```
# Analyze cosine similarities
# ... (Identify similar/dissimilar data points, visualize relationships)

# Jaccard similarity matrix (sample)
print("Sample of Jaccard similarity matrix:")
print(pd.DataFrame(jaccard_dists[:5, :5], columns=X_preprocessed.columns))

# Analyze Jaccard similarities
# ... (Cluster data points based on similarities, identify overlapping features)

# Applications (e.g., clustering, nearest neighbor search)
# ... (Integrate distance matrices into relevant algorithms)
```

Week 7: Linear Regression Models and Applications

Aim:

- Implement linear regression models to predict continuous target variables.
- Interpret model coefficients and assess goodness of fit.

Program:

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

# Load and pre-process your data (similar to Week 1)
# ...

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y_target,
test_size=0.2)

# Create a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on test data
predictions = model.predict(X_test)

# Evaluate model performance (metrics like R-squared, mean squared error)
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
# Interpret coefficients (variable importance)
coefficients = model.coef_
importance = pd.DataFrame({"Feature": X_train.columns, "Importance": coefficients})
print(importance.sort_values(by="Importance", ascending=False))

# Visualize predicted vs. actual values (optional)
# ... (Use libraries like matplotlib or seaborn to create scatter plots)

# Applications (e.g., forecasting, anomaly detection)
# ... (Implement the model for specific application purposes)
```

Week 8: Logistic Regression Models and Applications

Aim:

- Implement logistic regression models to predict binary or categorical target variables.
- Interpret model coefficients and evaluate model performance for classification tasks.

Program:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.model_selection import train_test_split

# Load and pre-process your data (similar to Week 1, handling categorical variables)
# ...

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y_target, test_size=0.2)

# Create a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on test data
predictions = model.predict(X_test)

# Evaluate model performance (accuracy, precision, recall, F1-score, ROC AUC score)
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
```

```
f1 = f1_score(y_test, predictions)
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC AUC score:", roc_auc)

# Interpret coefficients (variable importance)
coefficients = model.coef_.flatten()
importance = pd.DataFrame({"Feature": X_train.columns, "Importance": coefficients})
print(importance.sort_values(by="Importance", ascending=False))

# Visualize confusion matrix and/or ROC curve (optional)
# ... (Use libraries like matplotlib or seaborn)

# Applications (e.g., risk assessment, fraud detection, sentiment analysis)
# ... (Implement the model for specific application purposes)
```

Week 9: Support Vector Machines (SVM) Model and Applications

Aim:

- Implement Support Vector Machines (SVMs) for classification and regression tasks.
- Explore different kernel functions and hyperparameter tuning to optimize performance.
- Understand the underlying principle of SVMs and their decision boundaries.

Program:

```
from sklearn.svm import SVC, SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error

# Load and pre-process your data (similar to Week 1, handling scaling for SVM)
# ...

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y_target,
test_size=0.2)

# Classification

# Create an SVM classifier with kernel functions and hyperparameter tuning
# Example with linear kernel and GridSearchCV for hyperparameter optimization
from sklearn.model_selection import GridSearchCV
params_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
svc_linear = SVC(kernel='linear')
clf = GridSearchCV(svc_linear, params_grid, cv=5)
clf.fit(X_train, y_train)
best_model = clf.best_estimator_
predictions = best_model.predict(X_test)

# Evaluate classification performance
accuracy = accuracy_score(y_test, predictions)
print("SVM (linear kernel) accuracy:", accuracy)

# Explore other kernel functions (e.g., RBF, polynomial) and hyperparameter
combinations
```

```
# ... (Repeat similar grid search for different kernels and hyperparameters)

# Regression

# Create an SVR model for regression (adjust parameters and evaluation metrics)
svr = SVR(kernel='linear')
svr.fit(X_train, y_train)
predictions = svr.predict(X_test)

# Evaluate regression performance
mse = mean_squared_error(y_test, predictions)
print("SVR (linear kernel) MSE:", mse)

# ... (Explore other kernels and hyperparameters for regression)

# Visualize decision boundaries (for classification)
# ... (Use libraries like scikit-learn's `plot_svc` or custom visualization techniques)
```

Week 10: Misclassification Rate on Hold-Out Test Set and Categorical/Continuous Target Evaluation

Aim:

- Evaluate the performance of your trained model using misclassification rate on a hold-out test set.
- Analyze the predictions for both categorical and continuous target variables.

Program:

```
from sklearn.metrics import accuracy_score, confusion_matrix,
mean_squared_error, classification_report

# Load and pre-process data (as in previous weeks)
# ...

# Split data into training, validation, and test sets
# (Consider holding out at least 20% for testing)
X_train, X_val, X_test, y_train, y_val, y_test = train_test_split(X_preprocessed,
y_target, test_size=0.2)

# Train your model (example using Logistic Regression)
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate misclassification rate on validation set for hyperparameter tuning
# (Optional: Perform grid search or randomized search if applicable)
predictions_val = model.predict(X_val)
val_accuracy = accuracy_score(y_val, predictions_val)
print("Validation accuracy:", val_accuracy)

# Evaluate misclassification rate on hold-out test set
predictions_test = model.predict(X_test)
test_accuracy = accuracy_score(y_test, predictions_test)
print("Test accuracy:", test_accuracy)
```

```
# Analyze predictions for categorical targets (Confusion matrix and classification
report)
if pd.api.types.is_categorical_dtype(y_target):
    cm = confusion_matrix(y_test, predictions_test)
    print("Confusion matrix:")
    print(cm)
    print("Classification report:")
    print(classification_report(y_test, predictions_test))

# Analyze predictions for continuous targets (Mean squared error)
elif pd.api.types.is_numeric_dtype(y_target):
    mse = mean_squared_error(y_test, predictions_test)
    print("Mean squared error:", mse)

# ... (Add other relevant evaluation metrics as needed)

# Further analysis
# - Visualize model performance (e.g., ROC curve for AUC score)
# - Compare performance with other models on the same dataset
# - Investigate misclassified instances to understand model limitations
```

Week 12: Traffic Light Control with Q-Learning

Program:

```
import numpy as np

class TrafficLight:
    def __init__(self):
        self.state = "red"
        self.timings = {"red": 5, "green": 3, "yellow": 2}

    def next_state(self):
        next_state = "red"
        if self.state == "red":
            next_state = "green"
        elif self.state == "green":
            next_state = "yellow"
        return next_state

    def update(self):
        self.timings[self.state] -= 1
        if self.timings[self.state] <= 0:
            self.state = self.next_state()

class Car:
    def __init__(self, light):
        self.position = "stop"
        self.light = light

    def move(self):
        if self.light.state == "green":
            self.position = "go"
        else:
            self.position = "stop"

class Environment:
    def __init__(self):
        self.light = TrafficLight()
        self.car = Car(self.light)

    def step(self):
        self.light.update()
```

```
        self.car.move()
        reward = 1 if self.car.position == "go" else -1
        return reward, self.get_state()

    def get_state(self):
        return (self.light.state, self.car.position)

# Q-learning implementation to learn optimal actions
# ... (Adapt Q-learning algorithm to this environment)

env = Environment()

# ... (Train and evaluate the Q-learning agent)
```