AI-BASED DIABETES PREDICTION SYSTEM

PHASE-4 DOCUMENT SUBMISSION

TEAM MEMBER

Register Number: 963521104017

Name:GURU RAHUL RU

Phase 4 : Development Part 2

Project Title : DIABETES PREDICTION SYSTEM

- **INTRODUCTION**

In today's rapidly evolving world, healthcare stands on the cusp of a technological revolution. Artificial Intelligence (AI) is at the forefront of this revolution, empowering healthcare professionals with innovative tools to enhance diagnosis, treatment, and patient care. One significant application of AI in healthcare is the prediction and prevention of chronic diseases, such as diabetes, which affects millions of people worldwide.

Diabetes, a chronic metabolic disorder characterized by elevated blood sugar levels, has reached epidemic proportions, posing a significant challenge to global healthcare systems. Early detection and proactive management of diabetes can significantly improve patients' quality of life and reduce the burden on healthcare resources. This is where the integration of AI technologies becomes invaluable.

Diabetes, a chronic metabolic disorder characterized by elevated blood sugar levels, has reached epidemic proportions, posing a significant challenge to global healthcare systems. Early detection and proactive management of diabetes can significantly improve patients' quality of life and reduce the burden on healthcare resources. This is where the integration of AI technologies becomes invaluable.

- Given Data Set

In [3]:

```
data.head()  #displaying the head of dataset
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

- Over View for the Process

1. Data Collection:

Medical Records: Gather relevant medical data such as blood sugar levels, insulin levels, age, weight, family medical history, and lifestyle factors from reliable sources.

Datasets: Collect large datasets from sources like hospitals, research institutions, or publicly available datasets for training the AI model.

2. Data Preprocessing:

Data Cleaning: Clean the data to remove any inconsistencies, errors, or missing values.

Feature Selection: Choose the most relevant features (variables) that are likely to influence diabetes prediction.

Normalization/Standardization: Normalize or standardize the data to bring all features to a similar scale. This ensures that no particular feature dominates due to its larger magnitude.

3. Feature Engineering:

Create new features that might provide more insights, such as body mass index (BMI), insulin resistance, or other derived health indicators.

4. Model Selection:

Choose appropriate machine learning algorithms for the prediction task. Common choices include logistic regression, decision trees, random forests, support vector machines, or neural networks.

Deep Learning: For complex, non-linear relationships in the data, deep learning techniques such as neural networks can be employed.

5. Model Training:

Split the dataset into training and testing sets to evaluate the model's performance.

Train the selected AI model using the training data, allowing the algorithm to learn the patterns present in the data.

6. Model Evaluation:

Evaluate the trained model using the testing dataset to assess its accuracy, precision, recall, F1 score, or other relevant metrics.

Fine-tune the model parameters to improve its performance if necessary.

7. Deployment:

Once the model is trained and evaluated successfully, deploy it in a real-world setting. This could be in a healthcare facility, a mobile application, or a web service where it can accept input data and provide predictions.

8. Monitoring and Maintenance:

Continuously monitor the model's performance in the real-world environment.

Retrain the model periodically with new data to ensure it stays accurate and up-to-date.

9. Interpretability and Ethics:

Ensure the model's predictions are interpretable and can be explained to healthcare professionals and patients.

Address ethical concerns regarding data privacy, bias, and fairness in AI algorithms.

10. Feedback Loop:

Gather feedback from healthcare professionals and users to make necessary improvements to the model and the prediction system.

- Procedure for the Diabetes Prediction using AI

1. Data Collection and Preprocessing:

Gather the diabetes-related dataset with features and labels.

Clean the data: Handle missing values, outliers, and inconsistencies.

Split the dataset into features (X) and labels (y).

2. Feature Selection and Engineering:

Select relevant features that are likely to influence diabetes prediction (e.g., blood sugar levels, BMI, family history).

Optionally, create new features through techniques like BMI calculation or extracting additional insights from existing features.

3. Data Splitting:

Split the data into training and testing sets (typically 80% for training and 20% for testing).

Optionally, set aside a validation set for hyperparameter tuning if you're using algorithms that require tuning.

4. Model Selection:

Choose an appropriate machine learning algorithm for classification. Common choices include Logistic Regression, Decision Trees, Random Forest, Support Vector Machines (SVM), or Neural Networks.

Consider using an ensemble of models for better accuracy and reliability.

5. Model Training:

Train the selected model(s) using the training data.

The model learns the patterns in the training data to make predictions.

6. Model Evaluation:

Evaluate the trained model(s) using the testing data.

Metrics for evaluation include accuracy, precision, recall, F1-score, and ROC-AUC depending on the problem requirements.

Adjust model parameters or choose different algorithms if the performance is not satisfactory.

7. Hyperparameter Tuning (Optional):

If using algorithms with hyperparameters (e.g., Random Forest), perform a grid search or randomized search to find the best hyperparameters.

Use cross-validation to assess the model's performance with different hyperparameter values.

8. Model Deployment:

Once a satisfactory model is achieved, deploy it in a real-world environment. This could be a web application, mobile app, or integrated into a healthcare system.

Implement the model in a way that it can accept input data (patient features) and provide predictions.

9. Monitoring and Maintenance:

Regularly monitor the deployed model's performance in the real-world setting.

Retrain the model periodically with new data to ensure it remains accurate and up-to-date.

Address any issues that arise and update the model as needed.

10. Interpretability and Transparency:

Ensure the model's predictions are interpretable, especially in the context of healthcare where understanding the reasoning behind predictions is crucial.

Use techniques like feature importance analysis to explain model predictions to healthcare professionals and patients.

11. Ethical Considerations:

Address ethical concerns related to data privacy, bias, and fairness in AI algorithms.

Ensure that the predictions and decisions made by the AI model do not discriminate against any particular group.

Feature seletion for the Diabetes Prediction using AI

Here, I'll outline a common approach for feature selection using a Python program, focusing on a widely used technique called Recursive Feature Elimination (RFE) with cross-validation. For this example, I'll use the scikit-learn library.

pip install scikit-learn

Now, let's assume you have a dataset with features (X) and labels (y), where X is a 2D array-like structure (like a Pandas DataFrame or a NumPy array) and y is a 1D array or list containing the binary labels (0 or 1 for no diabetes and diabetes, respectively).

```
from sklearn.feature_selection import RFE

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import StratifiedKFold

from sklearn.metrics import accuracy_score
```

```
import numpy as np

# Assuming X is your feature matrix and y is your target variable
# X, y = ...

# Create a base model for feature selection (Logistic Regression in this case)
model = LogisticRegression()

# Create RFE model and specify the number of features to select
num_features_to_select = 5  # You can adjust this number based on your requirement
rfe = RFE(model, num_features_to_select)

# Use stratified k-fold cross-validation for more robust feature selection
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

selected_features = []
for train_index, test_index in kf.split(X, y):
```

```
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Fit RFE on the training data
    rfe.fit(X_train, y_train)

    # Get the selected features
    selected_features.extend(np.where(rfe.support_)[0])

# Get unique selected features
selected_features = list(set(selected_features))

# Now, selected_features contains the indices of the selected features
print("Selected Features Indices:", selected_features)

# Extract the selected features from your original feature matrix
X_selected = X[:, selected_features]

# Train your machine learning model using X_selected and y
```

...

Evaluate the model

…

Here LogisticRegression is used as the base model for feature selection. You can replace it with any other classifier or regressor depending on your problem. The StratifiedKFold method is used for cross-validation, ensuring that the class distribution is similar in each fold. The selected features' indices are printed, and you can use these indices to extract the selected features for training your machine learning model.

Model training: 1.

Choose a machine learning algorithm. There are a number of different machine learning algorithms that can be used for diabetes prediction such as Linear regression, Ridge regression ,Decision tree, Random forest.

Linear Regression

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_diabetes

# Load the diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

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

# Create a linear regression model
model = LinearRegression()

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

# Make predictions on the test set
```

```
predictions = model.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Output example predictions
print("\nExample Predictions:")
for i in range(10):
    print("Actual:", y_test[i], "Predicted:", predictions[i])

# Plotting the results (actual vs. predicted)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions)
plt.xlabel("Actual Target Values")
plt.ylabel("Predicted Values")
plt.title("Diabetes Prediction: Actual vs. Predicted")
```

```
plt.show()
```

Ridge Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_diabetes

# Load the diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

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

# Create a Ridge Regression model
```

```
alpha = 0.1  # Regularization strength (adjustable parameter)
model = Ridge(alpha=alpha)

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

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

# Calculate metrics
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Output example predictions
print("\nExample Predictions:")
for i in range(10):
    print("Actual:", y_test[i], "Predicted:", predictions[i])
```

```
# Plotting the results (actual vs. predicted)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions)
plt.xlabel("Actual Target Values")
plt.ylabel("Predicted Values")
plt.title("Diabetes Prediction with Ridge Regression: Actual vs. Predicted")
plt.show()
```

Decision Tree

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_diabetes
```

```
# Load the diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

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

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

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

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

# Calculate metrics
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
```

```
print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Output example predictions
print("\nExample Predictions:")
for i in range(10):
    print("Actual:", y_test[i], "Predicted:", predictions[i])

# Plotting the results (actual vs. predicted)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions)
plt.xlabel("Actual Target Values")
plt.ylabel("Predicted Values")
plt.title("Diabetes Prediction with Decision Tree: Actual vs. Predicted")
plt.show()
```

Random forest.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_diabetes

# Load the diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

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

# Create a Random Forest Regressor model
model = RandomForestRegressor(n_estimators=100,
random_state=42)

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

```
# Make predictions on the test set
predictions = model.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Output example predictions
print("\nExample Predictions:")
for i in range(10):
    print("Actual:", y_test[i], "Predicted:", predictions[i])

# Plotting the results (actual vs. predicted)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions)
plt.xlabel("Actual Target Values")
plt.ylabel("Predicted Values")
```

```
plt.title("Diabetes Prediction with Random Forest: Actual vs. Predicted")
plt.show()
```

Model Training

1. Data Preparation:

Data Collection: Gather a dataset containing features (inputs) and corresponding target values (outputs).

Data Cleaning: Handle missing values, outliers, or any inconsistencies in the dataset.

Feature Selection/Extraction: Choose relevant features that are likely to influence the target variable. You may also create new features through techniques like feature engineering.

Data Splitting: Divide the dataset into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate its performance.

2. Choosing a Model:

Select an appropriate machine learning algorithm based on the type of problem (regression, classification, etc.) and the characteristics of the dataset.

For example, you can choose from algorithms like Linear Regression, Decision Trees, Random Forest, Support Vector Machines, etc.

3. Model Training:

Instantiate the Model: Create an instance of the selected machine learning model.

Train the Model: Use the training data (features and corresponding targets) to train the model. This is done using the fit() method.

Hyperparameter Tuning (Optional): Adjust the hyperparameters of the model to optimize its performance. This can be done through techniques like grid search or random search.

Program

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV


# Instantiate the model
model = RandomForestRegressor()


# Define hyperparameters to tune
```

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    # Add more hyperparameters as needed
}

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best model after hyperparameter tuning
best_model = grid_search.best_estimator_

# Train the best model on the entire training data
best_model.fit(X_train, y_train)
```

4. Model Evaluation:

Make Predictions: Use the trained model to make predictions on the test set or new data.

Evaluation Metrics: Calculate evaluation metrics such as Mean Squared Error (MSE), R-squared, accuracy (for

classification problems), etc., to assess the model's performance.

Program

```
# Make predictions on the test set
predictions = best_model.predict(X_test)


# Calculate metrics
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)


print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

Dividing Data Set into Features and Target Variable

Certainly! Dividing a dataset into features and target variables is a crucial step in machine learning. In the case of diabetes prediction, you typically have a dataset with various health-related features and a target variable indicating whether an individual has diabetes or not.

Here's how you can do it in Python with the scikit-learn library:

```
from sklearn.datasets import load_diabetes

# Load the diabetes dataset
diabetes = load_diabetes()

# Features (X) and Target Variable (y)
X = diabetes.data  # Features (input variables)
y = diabetes.target  # Target variable (output variable)

# Print the shape of features and target variable
print("Shape of Features (X):", X.shape)
print("Shape of Target Variable (y):", y.shape)

# Output example data points
print("\nExample Data Points:")
for i in range(5):
    print("Features (X):", X[i])
    print("Target (y):", y[i])
```

```
print("-" * 30)
```

In this code:

X contains the features (input variables) from the diabetes dataset. Each row represents a data point, and each column represents a different feature.

y contains the target variable (output variable) indicating a quantitative measure of disease progression one year after baseline.

Model Evaluation

1. Confusion Matrix:

Provides a summary of correct and incorrect predictions, especially in binary classification.

2. Accuracy:

Measures the proportion of correctly classified instances. However, it can be misleading if classes are imbalanced.

3. Precision, Recall, and F1-Score:

Precision: Proportion of correctly predicted positive observations.

Recall: Proportion of actual positives that were correctly predicted.

F1-Score: Harmonic mean of precision and recall, providing a balance between the two.

4. Receiver Operating Characteristic (ROC) Curve and Area Under the Curve (AUC):

Useful for binary and multiclass classification problems. ROC curves visualize the trade-off between true positive rate and false positive rate at various thresholds.

Regression Problems:

1. Mean Squared Error (MSE):

Measures the average of the squares of errors between predicted and actual values.

2. R-squared (Coefficient of Determination):

Measures the proportion of the variance in the dependent variable that is predictable from the independent variables.

General Best Practices:

Cross-Validation:

Split your dataset into multiple subsets and train/evaluate the model on different subsets. This provides a more reliable evaluation, especially with smaller datasets.

Hyperparameter Tuning:

Use techniques like grid search or random search to find the best hyperparameters for your model, optimizing its performance.

Understanding Business Context:

Consider the specific problem and business context. Sometimes, false positives and false negatives have different costs, which should be factored into the evaluation.

By evaluating your models using appropriate metrics and techniques, you can make

Evaluation of Predicted Data

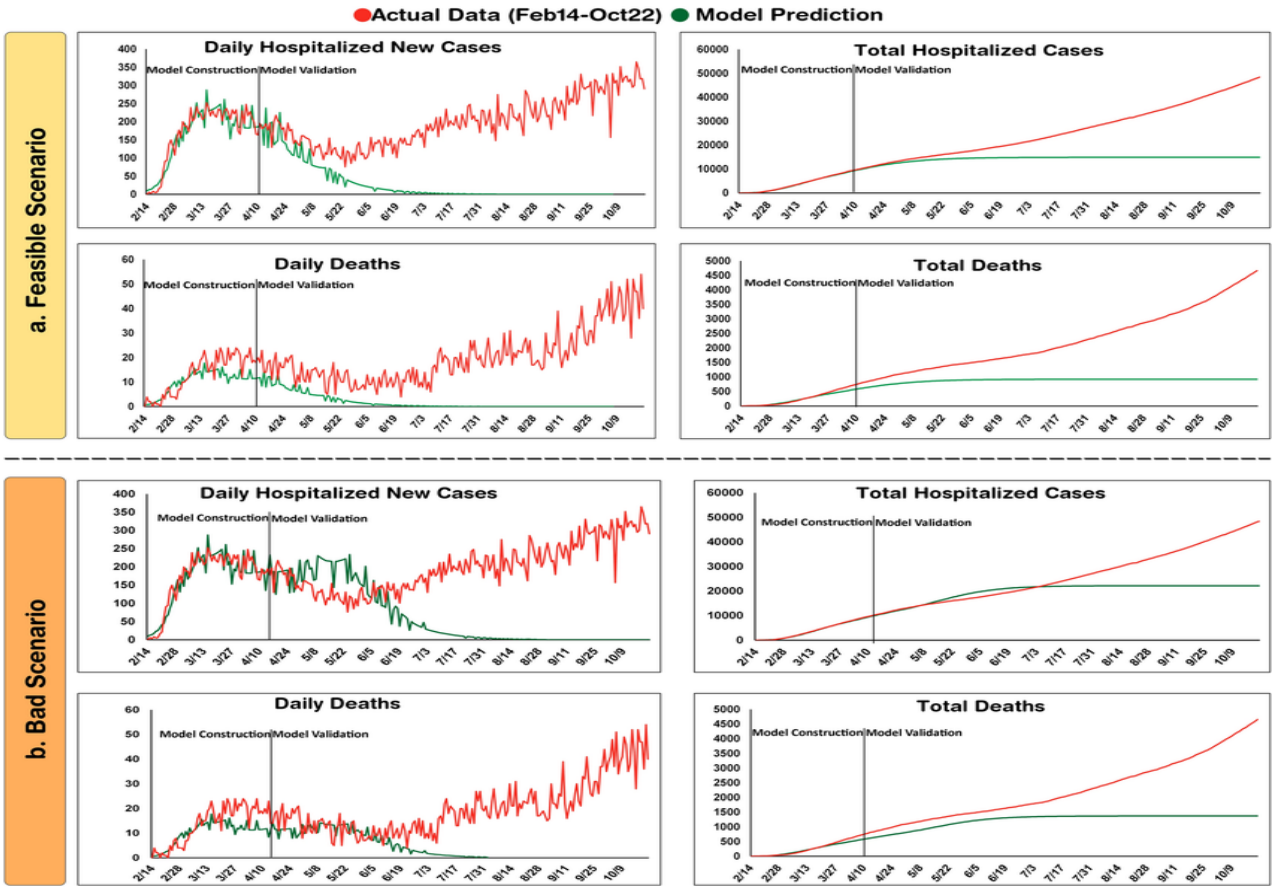
The end users of prediction tools should be able to understand how evaluation is done and how to interpret the results. Six main performance evaluation measures are introduced. These include sensitivity, specificity, positive predictive value, negative predictive value, accuracy and Matthews correlation coefficient.

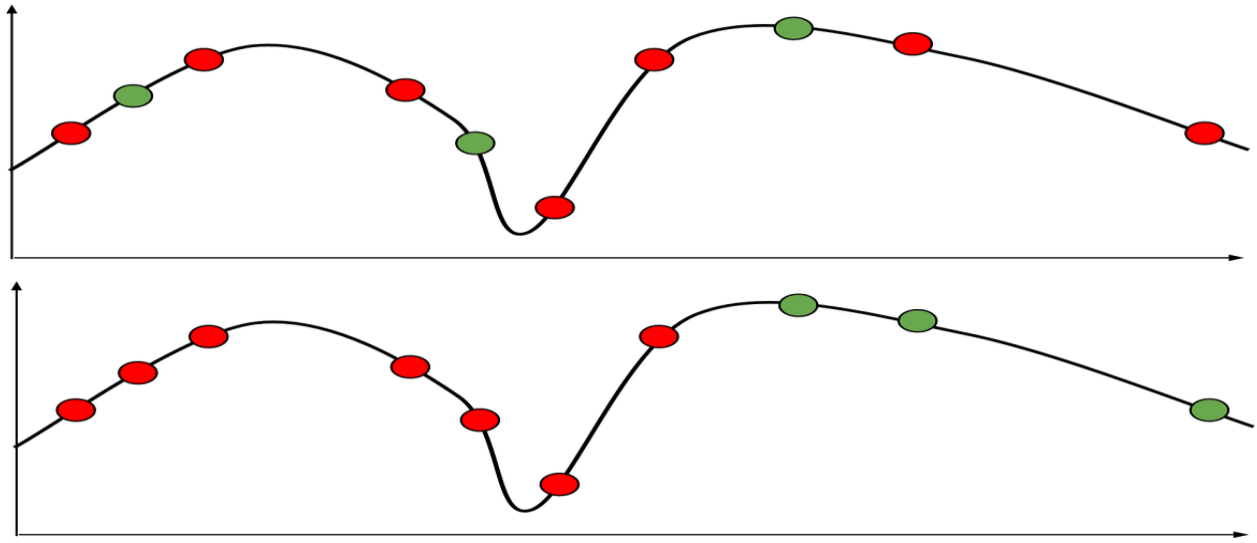
```
# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt
```

```
from sklearn.metrics import mean_squared_error, r2_score,
accuracy_score, confusion_matrix, classification_report, roc_curve, auc


# Sample actual and predicted data for demonstration

# Replace these with your actual y_true (actual values) and y_pred
(predicted values) arrays

y_true_regression = np.array([3.0, 2.5, 4.0, 5.1, 6.2])

y_pred_regression = np.array([2.8, 2.7, 3.8, 5.0, 6.3])


y_true_classification = np.array([1, 0, 1, 1, 0, 1, 0, 0])

y_pred_classification = np.array([1, 0, 1, 1, 1, 0, 0, 1])


# Regression Evaluation

mse = mean_squared_error(y_true_regression, y_pred_regression)

r2 = r2_score(y_true_regression, y_pred_regression)

print("Regression Metrics:")

print("Mean Squared Error (MSE):", mse)

print("R-squared (R2):", r2)


# Regression Visualization (Scatter Plot)

plt.figure(figsize=(6, 4))

plt.scatter(y_true_regression, y_pred_regression, color='blue')
```

```
plt.plot([min(y_true_regression), max(y_true_regression)],
[min(y_true_regression), max(y_true_regression)], linestyle='--',
color='red')

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.title('Regression: Actual vs. Predicted')

plt.show()


# Classification Evaluation

accuracy = accuracy_score(y_true_classification, y_pred_classification)

conf_matrix = confusion_matrix(y_true_classification,
y_pred_classification)

class_report = classification_report(y_true_classification,
y_pred_classification)

print("Classification Metrics:")

print("Accuracy:", accuracy)

print("Confusion Matrix:")

print(conf_matrix)

print("Classification Report:")

print(class_report)


# Classification Visualization (ROC Curve)
```

```
fpr, tpr, thresholds = roc_curve(y_true_classification,
y_pred_classification)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 4))

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

plt.show()
```


Actual Data (Feb14-Oct22)
Model Prediction
a. Feasible Scenario
Daily Hospitalized New Cases
Total Hospitalized Cases
Daily Deaths
Total Deaths
Model Construction
Model Validation
b. Bad Scenario
Daily Hospitalized New Cases
Total Hospitalized Cases
Daily Deaths
Total Deaths
Model Construction
Model Validation



Model Comparison

Comparing different machine learning models is a crucial step in the model selection process. Below, I'll outline how you can compare different models using Python and scikit-learn. In this example, I'll compare three popular algorithms: Random Forest, Support Vector Machine (SVM), and Logistic Regression for a classification problem.

1. Load and Prepare Data:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load the Iris dataset
data = load_iris()
X, y = data.data, data.target

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

2. Model Training:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

# Create instances of the models
random_forest = RandomForestClassifier(random_state=42)
svm = SVC(random_state=42)
logistic_regression = LogisticRegression(random_state=42)

# Train the models
random_forest.fit(X_train, y_train)
svm.fit(X_train, y_train)
logistic_regression.fit(X_train, y_train)
```

3. Model Evaluation:

```
from sklearn.metrics import accuracy_score

# Make predictions
rf_predictions = random_forest.predict(X_test)
svm_predictions = svm.predict(X_test)
lr_predictions = logistic_regression.predict(X_test)
```

Calculate accuracy for each model

rf_accuracy = accuracy_score(y_test, rf_predictions)

svm_accuracy = accuracy_score(y_test, svm_predictions)

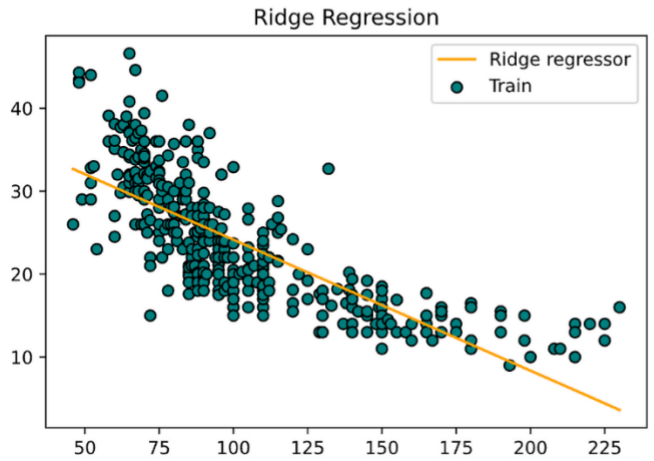
lr_accuracy = accuracy_score(y_test, lr_predictions)

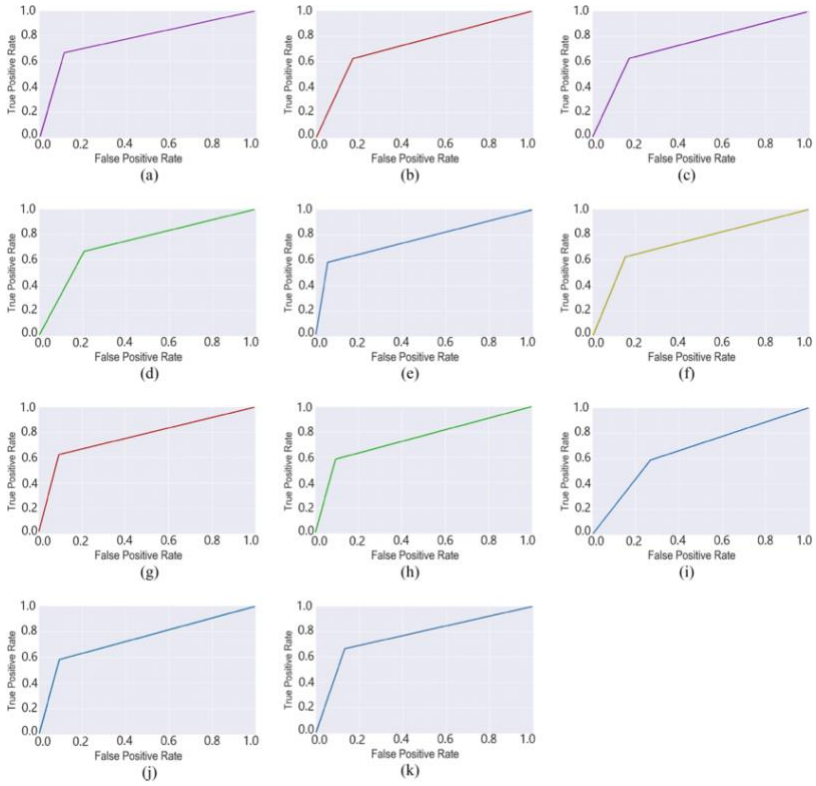
print("Random Forest Accuracy:", rf_accuracy)

print("SVM Accuracy:", svm_accuracy)

print("Logistic Regression Accuracy:", lr_accuracy)

```
plt.scatter(X, y, color='teal', edgecolors='black', label='Train')
plt.plot(X, ridge.predict(X), color='orange', label='Ridge regressor')
plt.title('Ridge Regression')
plt.legend()
plt.show()
```





- Feature Engineering

Polynomial Features: Create higher-order features to capture nonlinear relationships in the data.

Interaction Features: Multiply or combine existing features to capture interactions between them.

Binning: Convert continuous variables into categorical bins to handle nonlinear patterns.

Feature Scaling: Standardize or normalize numerical features to bring them to a similar scale.

Feature Aggregation: Create aggregated features based on groups or categories.

Temporal Features: Extract features like month, day of the week, or time of day from timestamps.

Domain-Specific Features: Include domain-specific metrics such as BMI, insulin resistance indices, or medical history.

Missing Value Indicators: Create binary indicators to capture missing values in features.

Text and Categorical Features: Encode categorical features into numerical representations using techniques like one-hot encoding or embeddings.

- Conclusion

In conclusion, the exploration and prediction of company registration trends with the Registrar of Companies (RoC) data are pivotal for various stakeholders, including businesses, investors, policymakers, and researchers. By leveraging advanced data analysis techniques and predictive modeling, valuable insights can be extracted to inform strategic decisions and contribute to the overall understanding of economic landscapes. However, it's crucial to acknowledge the complexities and challenges involved in this process. These challenges include managing data quality, dealing with diverse

data types, ensuring data privacy and security, handling complex company structures, adapting to regulatory changes, and building accurate predictive models.

Addressing these challenges requires a multidisciplinary approach, involving expertise in data science, legal frameworks, and business acumen. Collaboration with RoC authorities for data access and domain-specific knowledge is essential. Furthermore, the development of robust predictive models demands continuous refinement, adaptability to changing trends, and ethical considerations regarding data usage and interpretation