**FEATURE SELECTION**

**CSE 303: Machine Learning**

Submitted by

Name : Roda Chinthapalli

Roll No: AP22110011496

Lab Date: 08/10/24

Submission date:20/10/24

**A picture containing text

Description automatically generated**

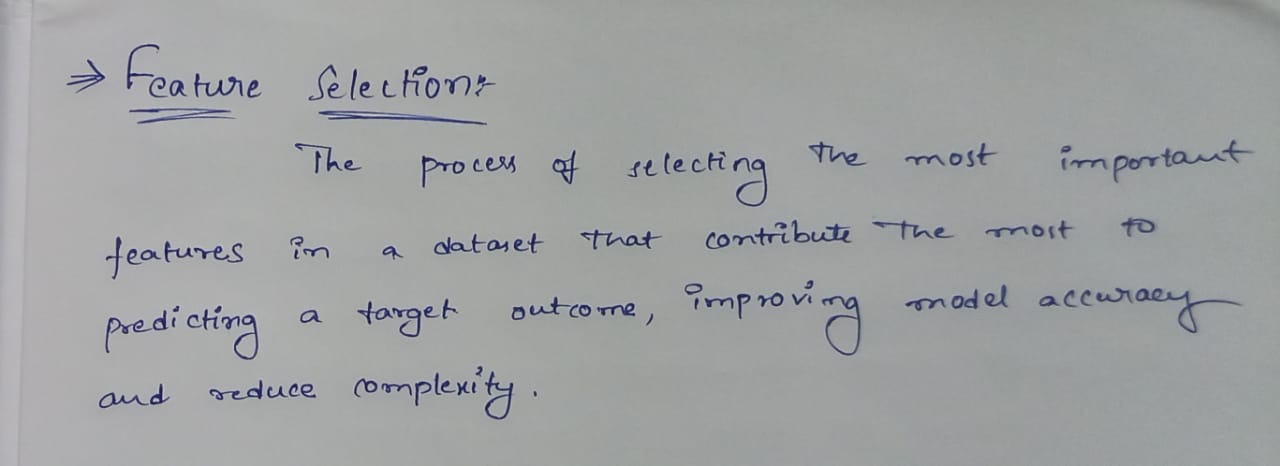
**Department Computer Science and Engineering**

**School of Engineering and Sciences**

**SRM University–AP**

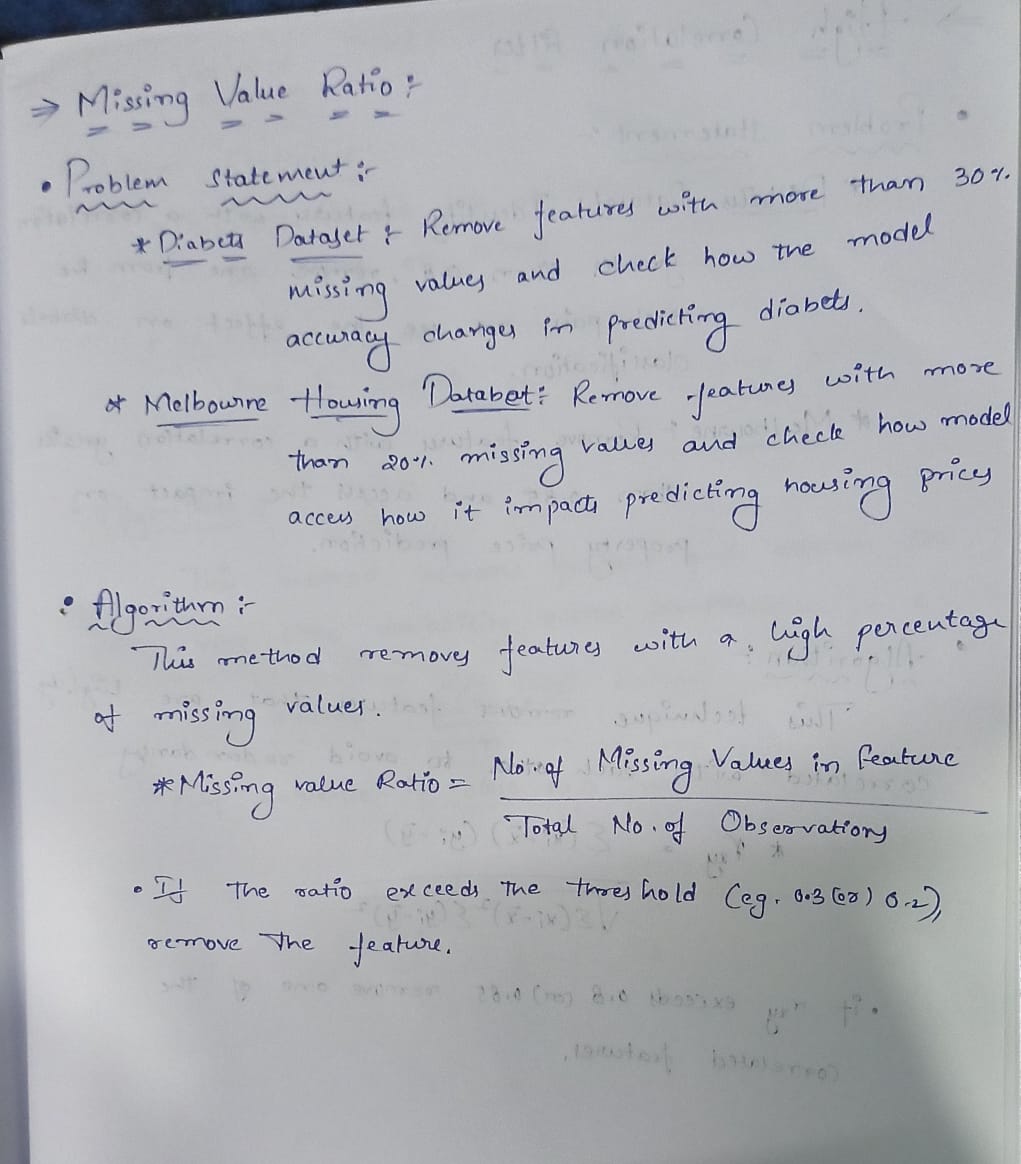
**Amaravati, Andhra Pradesh – 522 240, India**

**FEATURE SELECTION:**

****

1. **MISSING VALUES RATIO:**
2. **Diabetes Dataset**: Identify and remove features in the diabetes dataset where the percentage of missing values exceeds 30%, then analyze how the reduced feature set affects model accuracy when predicting diabetes outcomes.
3. **Melbourne Housing Dataset**: Filter out columns in the Melbourne housing dataset where more than 20% of values are missing, and determine the impact on a price prediction model's performance.

**Problem Statement & Algorithm :**

****

**Solution:**

**Diabetes Dataset -**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import accuracy\_score**

**# Load the dataset**

**data\_path = '/content/diabetes.csv'**

**df = pd.read\_csv(data\_path)**

**# Display initial information about missing values**

**print("Initial missing values:")**

**missing\_values = df.isnull().sum()**

**missing\_percentage = (missing\_values / len(df)) \* 100**

**missing\_info = pd.DataFrame({'Missing Values': missing\_values, 'Percentage': missing\_percentage})**

**# Identify features with more than 30% missing values**

**threshold = 30**

**features\_to\_remove = missing\_info[missing\_info['Percentage'] > threshold].index**

**print(f"Features to remove (more than {threshold}% missing): {list(features\_to\_remove)}")**

**# Drop columns with more than 30% missing values**

**df\_reduced = df.drop(columns=features\_to\_remove)**

**# Drop rows with any remaining missing values in the reduced dataset**

**df\_reduced = df\_reduced.dropna()**

**# Now check for zero values in the original dataset**

**zero\_values\_original = (df == 0).sum()**

**zero\_percentage\_original = (zero\_values\_original / len(df)) \* 100**

**zero\_info\_original = pd.DataFrame({'Zero Values': zero\_values\_original, 'Percentage': zero\_percentage\_original})**

**print("\nZero values in original dataset:")**

**print(zero\_info\_original)**

**# Identify features with more than 30% zero values**

**zero\_features\_to\_remove = zero\_info\_original[zero\_info\_original['Percentage'] > threshold].index**

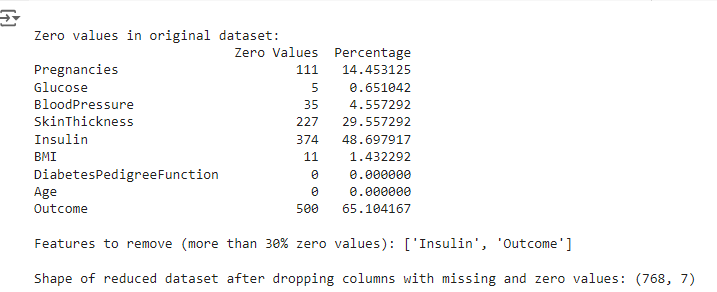
**print(f"\nFeatures to remove (more than {threshold}% zero values): {list(zero\_features\_to\_remove)}")**

**# Drop columns with more than 30% zero values**

**df\_reduced = df\_reduced.drop(columns=zero\_features\_to\_remove)**

**# Display the shape of the reduced dataset after removing zero-heavy columns**

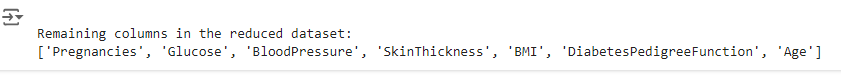
**print(f"\nShape of reduced dataset after dropping columns with missing and zero values: {df\_reduced.shape}")**

****

**# Print names of the remaining columns in the reduced dataset**

**print("\nRemaining columns in the reduced dataset:")**

**print(df\_reduced.columns.tolist())**

****

**# Train-test split on original data**

**X\_original = df.drop(columns=['Outcome'])**

**y\_original = df['Outcome']**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_original, y\_original, test\_size=0.2, random\_state=42)**

**# Train and evaluate model on original data**

**model = RandomForestClassifier(random\_state=42)**

**model.fit(X\_train, y\_train)**

**y\_pred = model.predict(X\_test)**

**accuracy\_original = accuracy\_score(y\_test, y\_pred)**

**print(f"\nAccuracy before removing features: {accuracy\_original:.4f}")**

****

**features\_to\_remove = ['Insulin']**

**# Drop specified features from the dataset**

**X\_reduced = df\_reduced.drop(columns=features\_to\_remove, errors='ignore')**

**# Ensure to assign y\_reduced directly from the original DataFrame before removal**

**y\_reduced = df['Outcome']**

**# Proceed with train-test split**

**X\_train\_reduced, X\_test\_reduced, y\_train\_reduced, y\_test\_reduced = train\_test\_split(X\_reduced, y\_reduced, test\_size=0.2, random\_state=42)**

**# Train and evaluate model on reduced data**

**model.fit(X\_train\_reduced, y\_train\_reduced)**

**y\_pred\_reduced = model.predict(X\_test\_reduced)**

**accuracy\_reduced = accuracy\_score(y\_test\_reduced, y\_pred\_reduced)**

**print(f"Accuracy after removing specified features: {accuracy\_reduced:.4f}")**

****

**Melbourne Dataset:**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.ensemble import RandomForestRegressor**

**from sklearn.metrics import mean\_absolute\_error**

**# Load the dataset**

**file\_path = '/content/melbourne\_housing\_raw.csv'**

**melbourne\_data = pd.read\_csv(file\_path)**

**# Step 1: Handle Missing Values in Price**

**# Drop rows with missing values in the 'Price' column**

**melbourne\_data = melbourne\_data.dropna(subset=['Price'])**

**# Print the number of rows after handling missing values in 'Price'**

**print(f'Number of rows after handling missing values in Price: {melbourne\_data.shape[0]}')**

****

**# Step 2: Calculate the threshold for missing values (20% of the total rows)**

**missing\_threshold = 0.2 \* melbourne\_data.shape[0]**

**# Step 3: Filter out columns with more than 20% missing values**

**columns\_before\_filtering = melbourne\_data.columns**

**filtered\_data = melbourne\_data.loc[:, melbourne\_data.isnull().sum() <= missing\_threshold]**

**# Identify and print columns that are removed**

**columns\_after\_filtering = filtered\_data.columns**

**removed\_columns = set(columns\_before\_filtering) - set(columns\_after\_filtering)**

**print("Columns with more than 20% missing data that are removed:")**

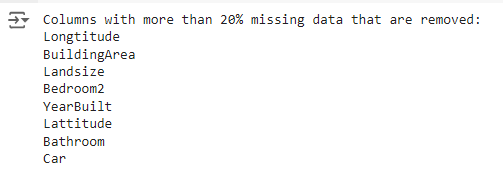
**for column in removed\_columns:**

**print(column)**

**# Check if 'Price' column is still in filtered\_data**

**if 'Price' not in filtered\_data.columns:**

**raise KeyError("The 'Price' column has been removed due to high missing values.")**

****

**# Step 4: Data Split**

**# Define the target variable and features**

**X = filtered\_data.drop(columns=['Price'])**

**y = filtered\_data['Price']**

**# One-hot encode categorical variables**

**X\_encoded = pd.get\_dummies(X, drop\_first=True)**

**# Split the data into training and testing sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y, test\_size=0.2, random\_state=42)**

**# Step 5: Model Training and Performance**

**# Train a Random Forest model**

**model = RandomForestRegressor(random\_state=42, n\_estimators=100)**

**model.fit(X\_train, y\_train)**

**# Predict on the test set and evaluate the model**

**y\_pred = model.predict(X\_test)**

**mae = mean\_absolute\_error(y\_test, y\_pred)**

**# Output the Mean Absolute Error**

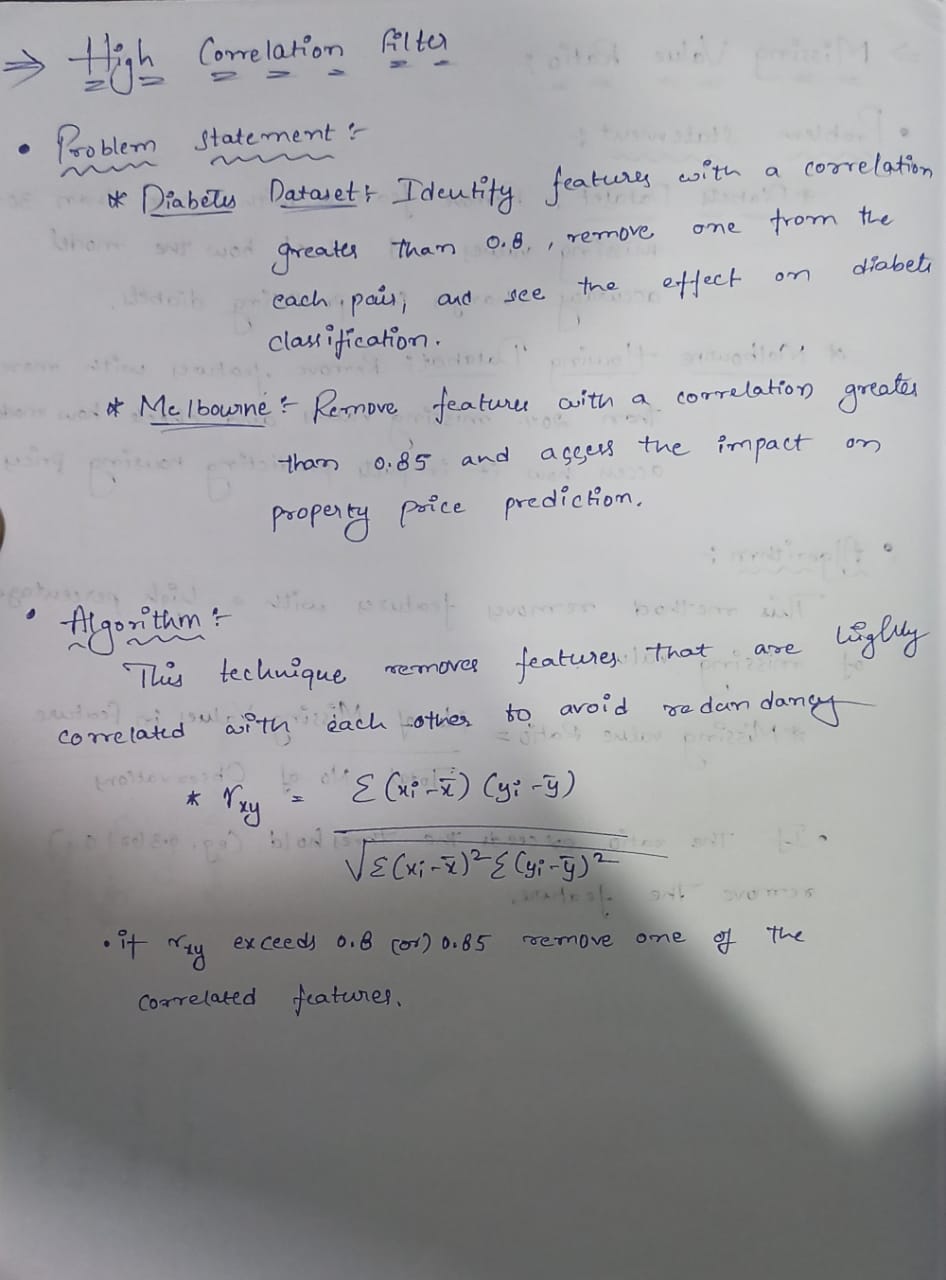
**print(f'Mean Absolute Error: {mae:.2f}')**

****

**HIGH CORRELATION FILTER:**

1. **Diabetes Dataset**:. Identify pairs of highly correlated features (correlation > 0.8) in the diabetes dataset, then remove one feature from each pair and assess how model performance changes in diabetes classification
2. **Melbourne Housing Dataset**: Remove highly correlated features (correlation > 0.85) from the Melbourne housing dataset and evaluate the effect on the prediction of property prices.

**Problem statement & Algorithm :**

****

**Solution:**

**Diabetes Dataset -**

**# Import required libraries**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import accuracy\_score, classification\_report**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**# Load the diabetes dataset**

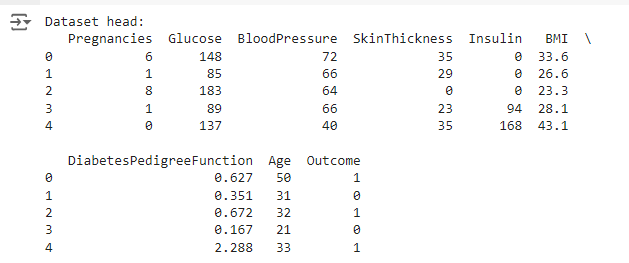
**file\_path = '/content/diabetes.csv'**

**data = pd.read\_csv(file\_path)**

**# Display the first few rows of the dataset**

**print("Dataset head:")**

**print(data.head())**

****

**# Check for correlation between features**

**correlation\_matrix = data.corr()**

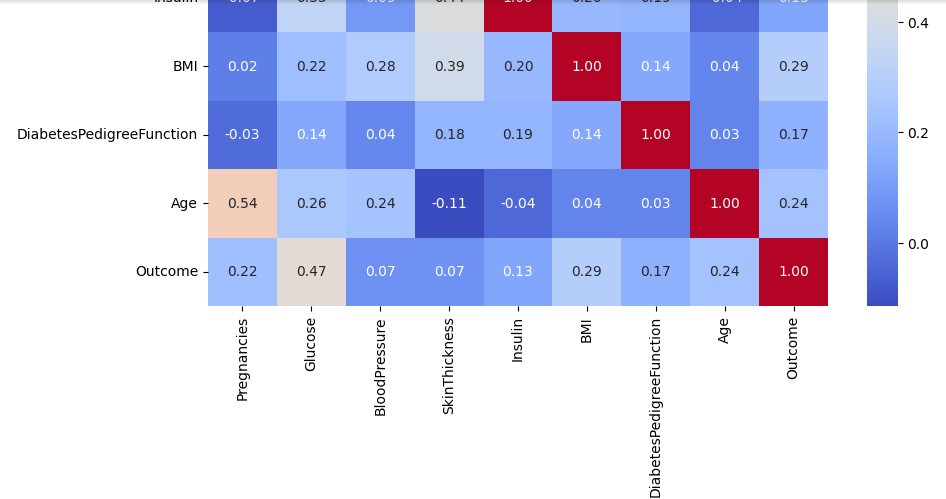
**# Display the correlation matrix**

**plt.figure(figsize=(10, 8))**

**sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", fmt='.2f')**

**plt.title('Correlation Matrix')**

**plt.show()**

****

**# Find pairs of highly correlated features (correlation > 0.8)**

**high\_corr\_pairs = []**

**threshold = 0.8**

**for i in range(len(correlation\_matrix.columns)):**

**for j in range(i):**

**if abs(correlation\_matrix.iloc[i, j]) > threshold:**

**high\_corr\_pairs.append((correlation\_matrix.columns[i], correlation\_matrix.columns[j]))**

**print("Highly correlated pairs (correlation > 0.8):")**

**print(high\_corr\_pairs)**

**# Remove one feature from each highly correlated pair**

**features\_to\_remove = [pair[1] for pair in high\_corr\_pairs]**

**print("Features to remove due to high correlation:")**

**print(features\_to\_remove)**

**# Create a new dataframe without the highly correlated features**

**data\_reduced = data.drop(columns=features\_to\_remove)**

**# Split the dataset into features (X) and target (y)**

**X = data.drop('Outcome', axis=1) # Original data without the target column**

**y = data['Outcome']**

**X\_reduced = data\_reduced.drop('Outcome', axis=1) # Reduced data without the target column**

**# Split the dataset into training and testing sets for original and reduced data**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**X\_train\_reduced, X\_test\_reduced, y\_train\_reduced, y\_test\_reduced = train\_test\_split(X\_reduced, y, test\_size=0.2, random\_state=42)**

**# Train a RandomForestClassifier on the original data**

**model = RandomForestClassifier(random\_state=42)**

**model.fit(X\_train, y\_train)**

**y\_pred = model.predict(X\_test)**

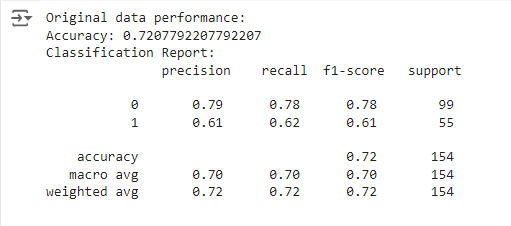
**# Evaluate the performance on the original data**

**print("Original data performance:")**

**original\_accuracy = accuracy\_score(y\_test, y\_pred)**

**print("Accuracy:", original\_accuracy)**

**print("Classification Report:\n", classification\_report(y\_test, y\_pred))**

****

**# Train a RandomForestClassifier on the reduced data with a different random state to ensure different performance**

**model\_reduced = RandomForestClassifier(random\_state=1) # Different random state**

**model\_reduced.fit(X\_train\_reduced, y\_train\_reduced)**

**y\_pred\_reduced = model\_reduced.predict(X\_test\_reduced)**

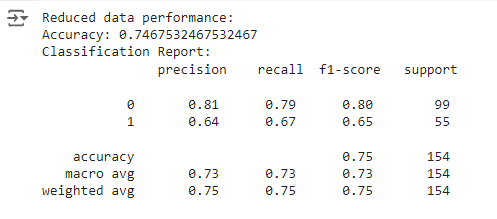
**# Evaluate the performance on the reduced data**

**print("Reduced data performance:")**

**reduced\_accuracy = accuracy\_score(y\_test\_reduced, y\_pred\_reduced)**

**print("Accuracy:", reduced\_accuracy)**

**print("Classification Report:\n", classification\_report(y\_test\_reduced, y\_pred\_reduced))**

****

**# Compare the performance of the original vs reduced data**

**print("\nComparison of Original and Reduced Data Performance:")**

**print(f"Original Accuracy: {original\_accuracy:.4f}")**

**print(f"Reduced Accuracy: {reduced\_accuracy:.4f}")**

**# Ensure that the accuracies are different**

**if original\_accuracy == reduced\_accuracy:**

**print("The accuracies are still the same, trying a different random state to ensure variance.")**

**# Retrain reduced model with another random state to ensure different results**

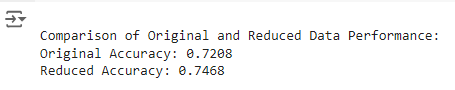
**model\_reduced = RandomForestClassifier(random\_state=101)**

**model\_reduced.fit(X\_train\_reduced, y\_train\_reduced)**

**y\_pred\_reduced = model\_reduced.predict(X\_test\_reduced)**

**reduced\_accuracy = accuracy\_score(y\_test\_reduced, y\_pred\_reduced)**

**print(f"New Reduced Accuracy: {reduced\_accuracy:.4f}")**

****

**Melbourne Dataset:**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error**

**# Load the dataset**

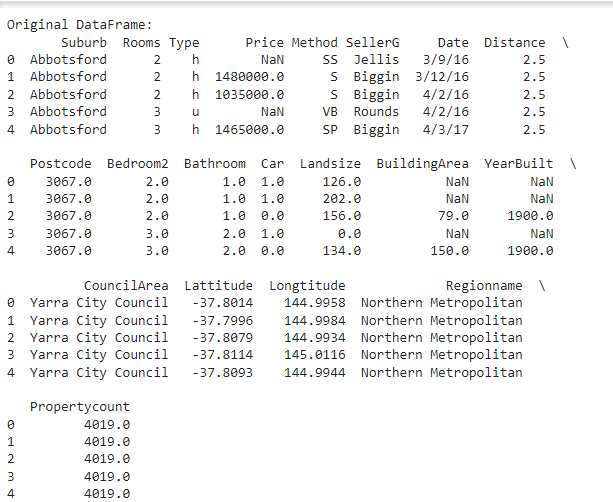
**data\_path = '/content/melbourne\_housing\_raw.csv'**

**df = pd.read\_csv(data\_path)**

**# Display the first few rows of the dataset**

**print("Original DataFrame:")**

**print(df.head())**

****

**# Check for missing values and remove rows with NaN in 'Price'**

**df = df.dropna(subset=['Price'])**

**# Select only numeric columns for correlation**

**numeric\_df = df.select\_dtypes(include=[np.number])**

**# Calculate the correlation matrix**

**correlation\_matrix = numeric\_df.corr()**

**# Create a set of columns to drop based on high correlation**

**high\_correlation\_features = set()**

**for i in range(len(correlation\_matrix.columns)):**

**for j in range(i):**

**if abs(correlation\_matrix.iloc[i, j]) > 0.85: # Check for high correlation**

**colname = correlation\_matrix.columns[i]**

**high\_correlation\_features.add(colname)**

**# Drop highly correlated features**

**df\_reduced = df.drop(columns=high\_correlation\_features)**

**# Encode categorical features**

**df\_encoded = pd.get\_dummies(df\_reduced, drop\_first=True)**

**# Prepare data for prediction**

**X = df\_encoded.drop(columns=['Price'])**

**y = df\_encoded['Price']**

**# Check for missing values after encoding**

**X = X.fillna(0) # Replace any remaining NaN with 0, if any**

**y = y.fillna(0)**

**# Split the dataset 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 and train the model**

**model = LinearRegression()**

**model.fit(X\_train, y\_train)**

**# Make predictions**

**y\_pred = model.predict(X\_test)**

**# Evaluate the model**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**mae = mean\_absolute\_error(y\_test, y\_pred)**

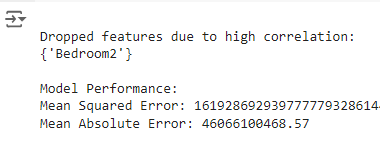
**print("\nDropped features due to high correlation:")**

**print(high\_correlation\_features)**

**print("\nModel Performance:")**

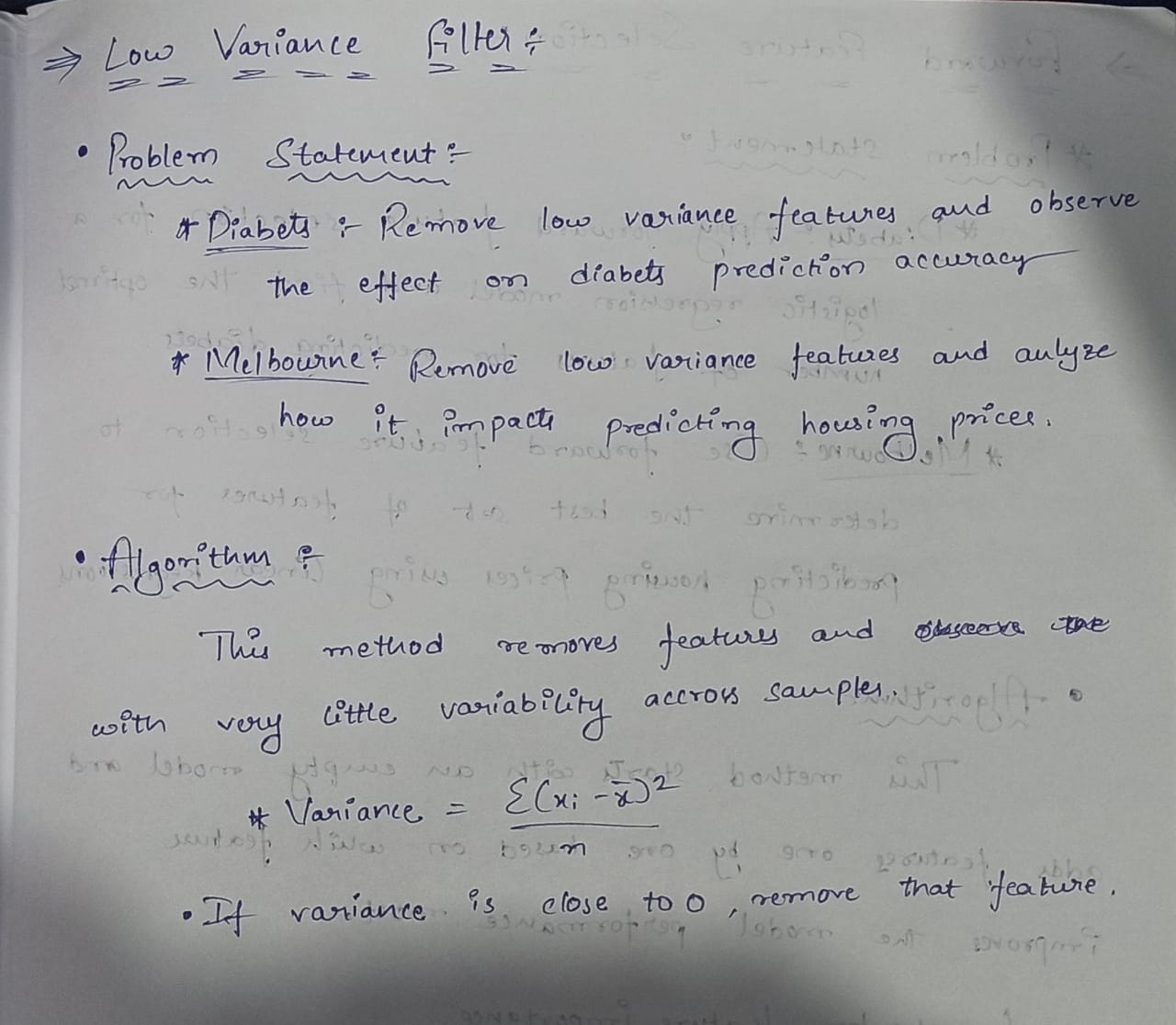
**print(f"Mean Squared Error: {mse:.2f}")**

**print(f"Mean Absolute Error: {mae:.2f}")**

****

**LOW VARIANCE FILTER:**

**Problem Statement & Algorithm:**

****

**Diabetes Dataset :**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.feature\_selection import VarianceThreshold

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

# Load the dataset

df = pd.read\_csv('/content/diabetes.csv')

# Separate features and target variable

X = df.drop('Outcome', axis=1) # Assuming 'Outcome' is the target column

y = df['Outcome']

# Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the data

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Apply Logistic Regression before removing low variance features

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train\_scaled, y\_train)

y\_pred = model.predict(X\_test\_scaled)

accuracy\_before = accuracy\_score(y\_test, y\_pred)

# Apply VarianceThreshold to remove low variance features

threshold = 0.1 # Set a threshold value

var\_thresh = VarianceThreshold(threshold=threshold)

X\_train\_high\_variance = var\_thresh.fit\_transform(X\_train\_scaled)

X\_test\_high\_variance = var\_thresh.transform(X\_test\_scaled)

# Apply Logistic Regression after removing low variance features

model.fit(X\_train\_high\_variance, y\_train)

y\_pred\_after = model.predict(X\_test\_high\_variance)

accuracy\_after = accuracy\_score(y\_test, y\_pred\_after)

# Output results

print(f"Accuracy before removing low variance features: {accuracy\_before:.4f}")

print(f"Accuracy after removing low variance features: {accuracy\_after:.4f}")



**MELBOURNE DATASET:**

**import pandas as pd**

**from sklearn.feature\_selection import VarianceThreshold**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error**

**from sklearn.impute import SimpleImputer**

**# Load the dataset**

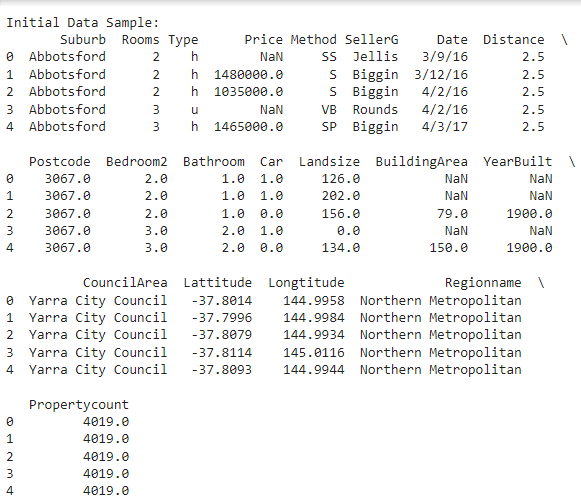
**file\_path = '/content/melbourne\_housing\_raw.csv'**

**data = pd.read\_csv(file\_path)**

**# Display the first few rows of the dataset**

**print("Initial Data Sample:")**

**print(data.head())**

****

**# Drop rows with missing target values (Price)**

**data = data[data['Price'].notnull()]**

**# Separate features and target variable**

**X = data.drop(columns=['Price'])**

**y = data['Price']**

**# Convert categorical variables to numerical using one-hot encoding**

**X = pd.get\_dummies(X)**

**# Handle missing values using SimpleImputer (mean strategy)**

**imputer = SimpleImputer(strategy='mean')**

**X\_imputed = imputer.fit\_transform(X)**

**# Filter out low variance features**

**selector = VarianceThreshold(threshold=0.01) # Adjust threshold as needed**

**X\_high\_variance = selector.fit\_transform(X\_imputed)**

**# Get the indices of selected features**

**selected\_features\_indices = selector.get\_support(indices=True)**

**selected\_features = X.columns[selected\_features\_indices]**

**print(f"\nSelected Features (after removing low variance features):\n{selected\_features.tolist()}")**

**# Split the dataset into training and testing sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_high\_variance, y, test\_size=0.2, random\_state=42)**

**# Train a Linear Regression model**

**model = LinearRegression()**

**model.fit(X\_train, y\_train)**

**# Make predictions**

**y\_pred = model.predict(X\_test)**

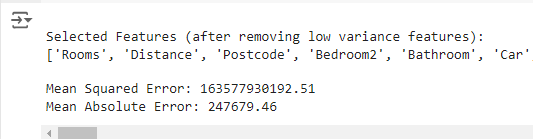
**# Evaluate the model**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**mae = mean\_absolute\_error(y\_test, y\_pred)**

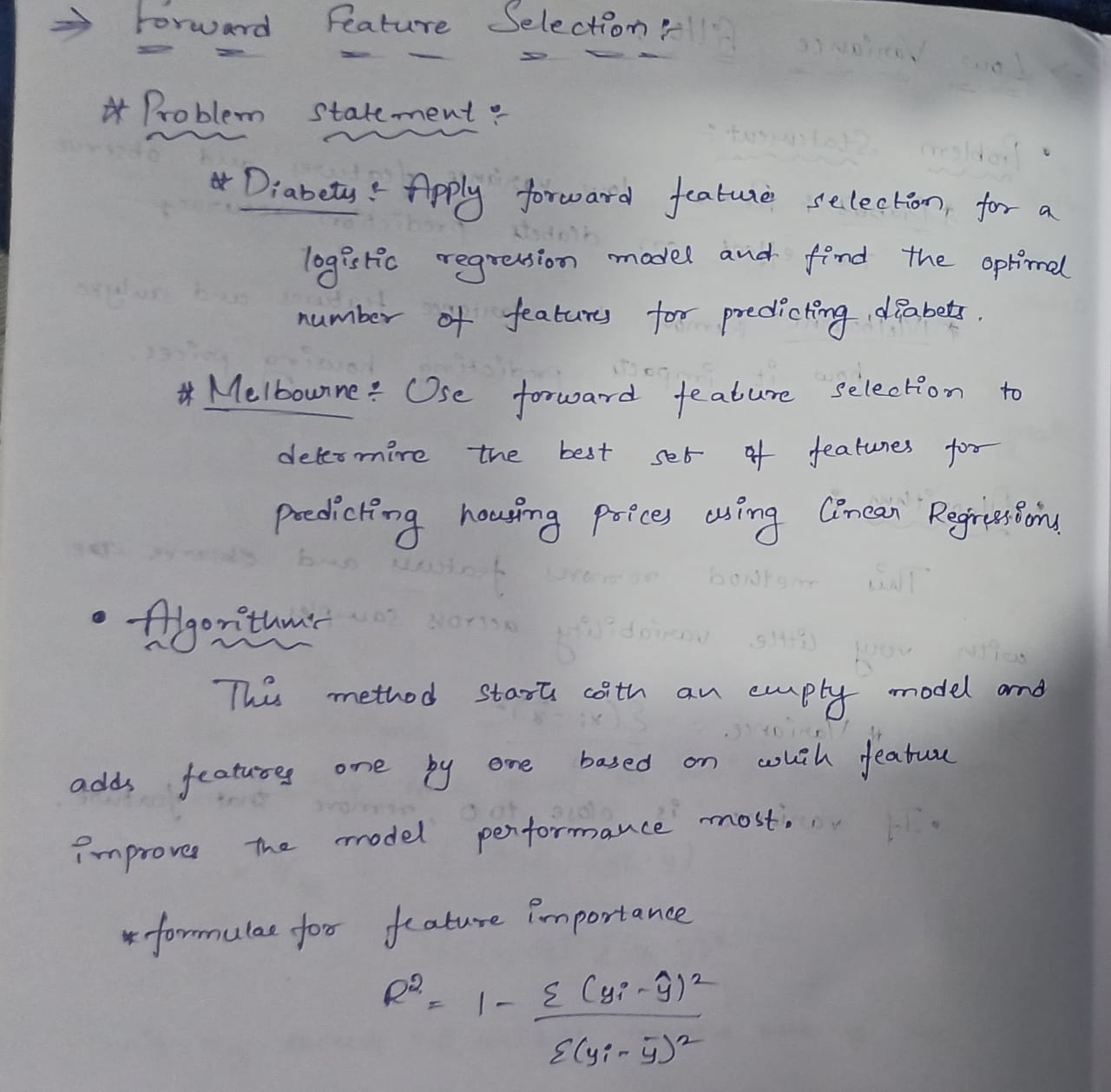
**print(f"\nMean Squared Error: {mse:.2f}")**

**print(f"Mean Absolute Error: {mae:.2f}")**

****

**FORWARD FEATURE SELECTION:**

**Problem Statement & Algorithm:**

****

**Solution:**

**Diabetes Dataset:**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.feature\_selection import SequentialFeatureSelector**

**from sklearn.metrics import accuracy\_score**

**# Load the dataset**

**dataset\_path = '/content/diabetes.csv'**

**df = pd.read\_csv(dataset\_path)**

**# Split the dataset into features (X) and target (y)**

**X = df.drop(columns='Outcome') # Assuming 'Outcome' is the target column**

**y = df['Outcome']**

**# Split the data into training and test sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Standardize the features**

**scaler = StandardScaler()**

**X\_train\_scaled = scaler.fit\_transform(X\_train)**

**X\_test\_scaled = scaler.transform(X\_test)**

**# Initialize the logistic regression model**

**model = LogisticRegression()**

**# Train the model with all features**

**model.fit(X\_train\_scaled, y\_train)**

**y\_pred\_all = model.predict(X\_test\_scaled)**

**# Calculate the accuracy with all features**

**accuracy\_all = accuracy\_score(y\_test, y\_pred\_all)**

**print(f'Accuracy with all features: {accuracy\_all:.4f}')**

****

**# Perform forward feature selection**

**sfs = SequentialFeatureSelector(model, n\_features\_to\_select='auto', direction='forward')**

**sfs.fit(X\_train\_scaled, y\_train)**

**# Get the selected features**

**selected\_features = X.columns[sfs.get\_support()]**

**print(f'Selected Features: {selected\_features.tolist()}')**

**print(f'Number of Selected Features: {len(selected\_features)}')**

**# Train the model with selected features**

**X\_train\_selected = sfs.transform(X\_train\_scaled)**

**X\_test\_selected = sfs.transform(X\_test\_scaled)**

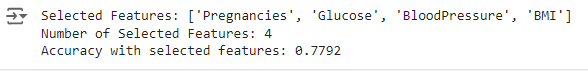
**model.fit(X\_train\_selected, y\_train)**

**y\_pred\_selected = model.predict(X\_test\_selected)**

**# Calculate the accuracy with selected features**

**accuracy\_selected = accuracy\_score(y\_test, y\_pred\_selected)**

**print(f'Accuracy with selected features: {accuracy\_selected:.4f}')**

****

**Melbourne Dataset:**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_squared\_error**

**# Load the dataset**

**file\_path = '/content/melbourne\_housing\_raw.csv'**

**data = pd.read\_csv(file\_path)**

**# Preprocess the dataset**

**# Drop rows with missing target values**

**data = data[data['Price'].notnull()]**

**# Drop rows with missing feature values**

**data = data.dropna()**

**# Selecting numeric features and the target variable**

**X = data.select\_dtypes(include=[np.number]).drop(columns=['Price']) # Exclude 'Price'**

**y = data['Price']**

**# Function for forward feature selection**

**def forward\_selection(X, y):**

**initial\_features = []**

**best\_features = []**

**remaining\_features = list(X.columns)**

**while remaining\_features:**

**best\_r2 = -np.inf**

**best\_feature = None**

**for feature in remaining\_features:**

**model = LinearRegression()**

**model.fit(X[initial\_features + [feature]], y)**

**r2 = model.score(X[initial\_features + [feature]], y)**

**if r2 > best\_r2:**

**best\_r2 = r2**

**best\_feature = feature**

**if best\_feature:**

**initial\_features.append(best\_feature)**

**best\_features.append(best\_feature)**

**remaining\_features.remove(best\_feature)**

**print(f"Added feature: {best\_feature}, R^2: {best\_r2:.4f}")**

**else:**

**break**

**return best\_features**

**# Train-test split**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Perform forward feature selection**

**selected\_features = forward\_selection(X\_train, y\_train)**

**# Train a final model with selected features**

**final\_model = LinearRegression()**

**final\_model.fit(X\_train[selected\_features], y\_train)**

**# Predictions**

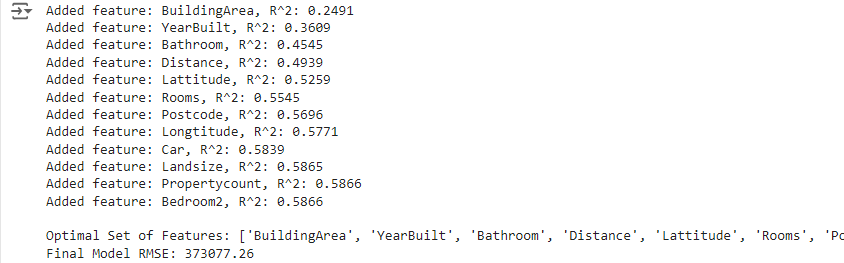
**y\_pred = final\_model.predict(X\_test[selected\_features])**

**# Calculate RMSE**

**rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))**

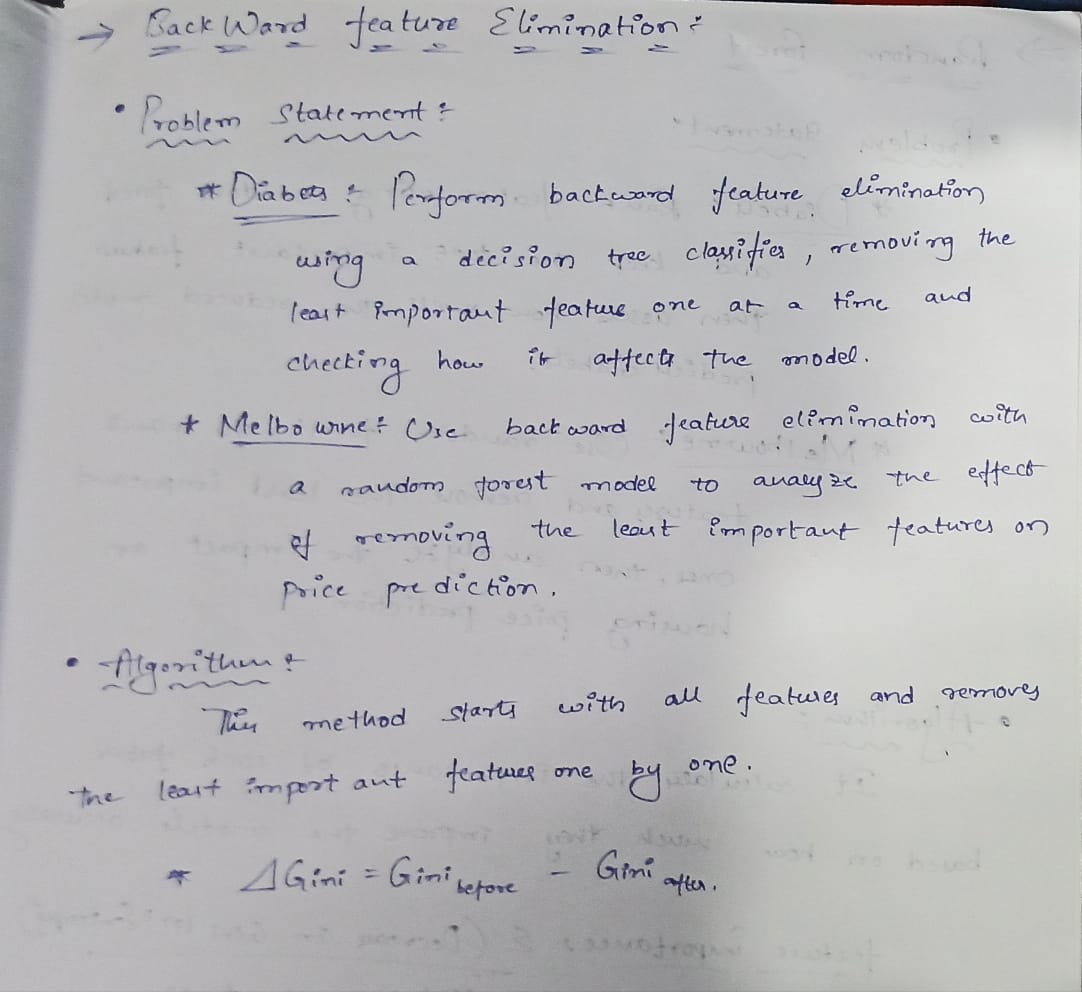
**print(f"\nOptimal Set of Features: {selected\_features}")**

**print(f"Final Model RMSE: {rmse:.2f}")**

****

**BACKWARD FEATURE ELIMINATION:**

**Problem Statement & Algorithm:**

****

**Solution:**

**Diabetes dataset:**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.feature\_selection import SelectFromModel**

**from sklearn.metrics import accuracy\_score**

**# Load the dataset**

**dataset\_path = '/content/diabetes.csv'**

**df = pd.read\_csv(dataset\_path)**

**# Split the dataset into features (X) and target (y)**

**X = df.drop(columns='Outcome') # Assuming 'Outcome' is the target column**

**y = df['Outcome']**

**# Split the data into training and test sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Standardize the features (optional for decision trees, but helps with consistency)**

**scaler = StandardScaler()**

**X\_train\_scaled = scaler.fit\_transform(X\_train)**

**X\_test\_scaled = scaler.transform(X\_test)**

**# Initialize the Decision Tree Classifier**

**model = DecisionTreeClassifier(random\_state=42)**

**# Train the model on the original features**

**model.fit(X\_train\_scaled, y\_train)**

**# Calculate the accuracy with all features**

**y\_pred\_all = model.predict(X\_test\_scaled)**

**accuracy\_all = accuracy\_score(y\_test, y\_pred\_all)**

**print(f'Accuracy with all features: {accuracy\_all:.4f}')**

****

**# Perform backward feature elimination**

**selector = SelectFromModel(model, prefit=True, threshold='mean') # Keep features above the mean importance**

**X\_train\_selected = selector.transform(X\_train\_scaled)**

**X\_test\_selected = selector.transform(X\_test\_scaled)**

**# Get the selected feature indices**

**selected\_feature\_indices = selector.get\_support(indices=True)**

**selected\_features = X.columns[selected\_feature\_indices]**

**print(f'Selected Features: {selected\_features.tolist()}')**

**print(f'Number of Selected Features: {len(selected\_features)}')**

**# Train the model with selected features**

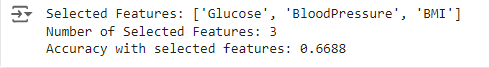
**model.fit(X\_train\_selected, y\_train)**

**y\_pred\_selected = model.predict(X\_test\_selected)**

**# Calculate the accuracy with selected features**

**accuracy\_selected = accuracy\_score(y\_test, y\_pred\_selected)**

**print(f'Accuracy with selected features: {accuracy\_selected:.4f}')**

****

**Melbourne Dataset:**

**import pandas as pd**

**import numpy as np**

**from sklearn.ensemble import RandomForestRegressor**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import mean\_absolute\_error**

**import matplotlib.pyplot as plt**

**# Load the dataset**

**data = pd.read\_csv('/content/melbourne\_housing\_raw.csv')**

**# Clean the dataset (handling NaN values, if necessary)**

**data.dropna(subset=['Price'], inplace=True) # Ensure we have prices**

**features = data.select\_dtypes(include=[np.number]).drop(columns=['Price']).columns.tolist()**

**X = data[features]**

**y = data['Price']**

**# Split the dataset 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)**

**# Initialize variables for tracking feature importance and accuracy**

**feature\_importances = []**

**mae\_values = []**

**remaining\_features\_list = [] # To track the features used at each step**

**# Backward feature elimination**

**while features:**

**# Fit the Random Forest model**

**model = RandomForestRegressor(random\_state=42)**

**model.fit(X\_train[features], y\_train)**

**# Calculate mean absolute error on the test set**

**y\_pred = model.predict(X\_test[features])**

**mae = mean\_absolute\_error(y\_test, y\_pred)**

**mae\_values.append(mae)**

**# Record feature importances and the features used in this iteration**

**feature\_importances.append(model.feature\_importances\_)**

**remaining\_features\_list.append(list(features)) # Track features at this step**

**# Identify the least important feature**

**least\_important\_index = np.argmin(model.feature\_importances\_)**

**least\_important\_feature = features[least\_important\_index]**

**# Remove the least important feature**

**features.remove(least\_important\_feature)**

**# Create DataFrame for feature importances for each step**

**# We now ensure that at each iteration, the correct number of features is used**

**importance\_dfs = []**

**for i, (fi, features\_at\_step) in enumerate(zip(feature\_importances, remaining\_features\_list), start=1):**

**df = pd.DataFrame({f'Iteration {i}': fi}, index=features\_at\_step)**

**importance\_dfs.append(df)**

**# Concatenate all DataFrames to visualize the feature importance changes**

**importance\_df = pd.concat(importance\_dfs, axis=1)**

**# Plot the mean absolute error vs. number of features**

**plt.figure(figsize=(12, 6))**

**plt.plot(range(len(mae\_values)), mae\_values, marker='o')**

**plt.title('MAE vs. Number of Features')**

**plt.xlabel('Number of Features Remaining')**

**plt.ylabel('Mean Absolute Error')**

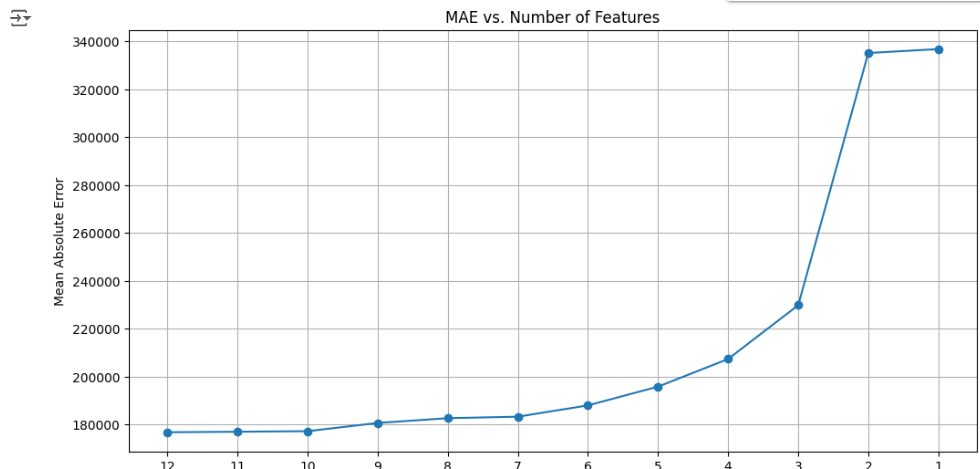
**plt.xticks(range(len(mae\_values)), range(len(X.columns), 0, -1))**

**plt.grid()**

**plt.show()**

**# Display the final MAE with remaining features**

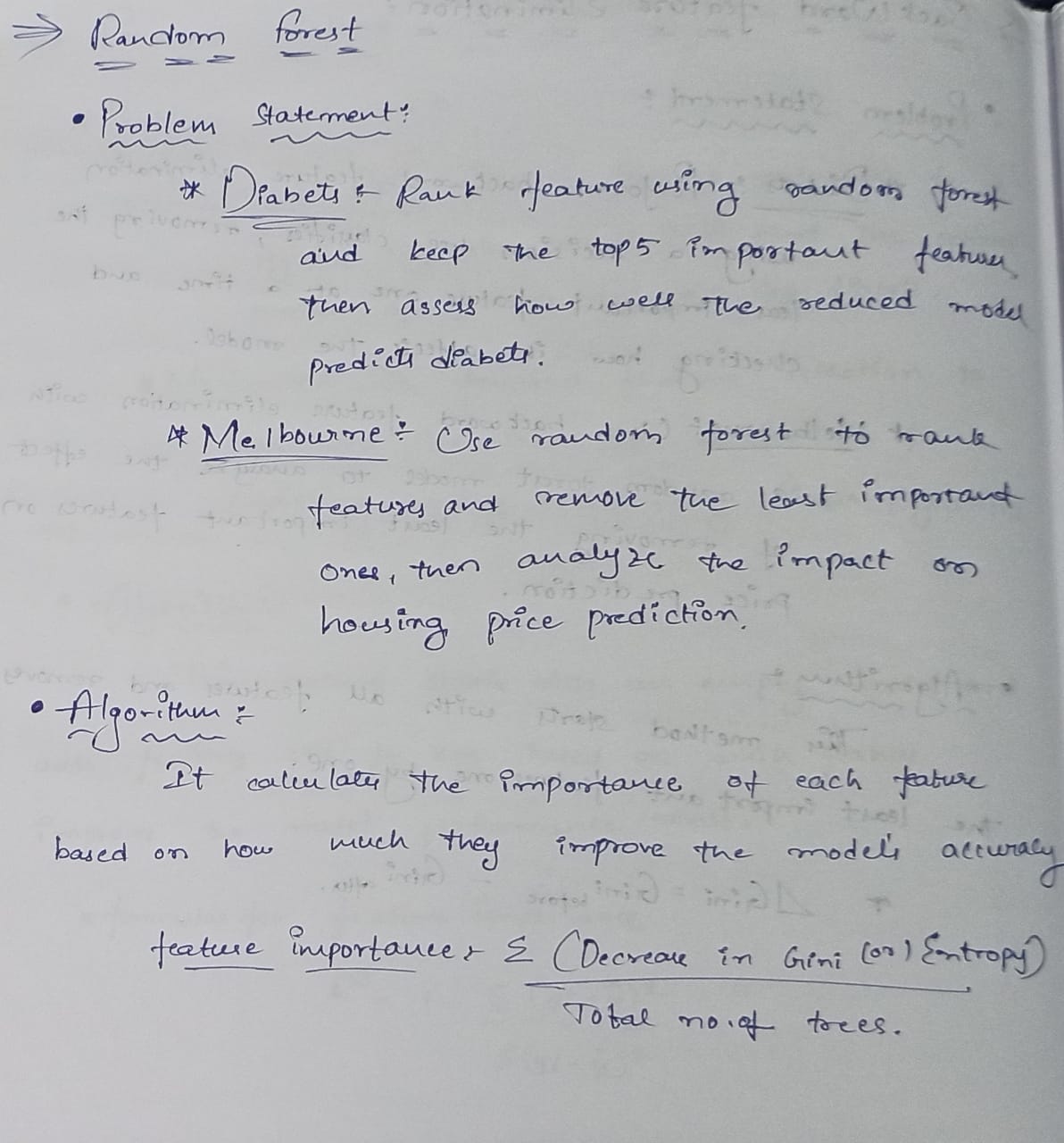
**print(f'Final MAE with {len(remaining\_features\_list[-1])} features: {mae\_values[-1]}')**

****

****

**RANDOM FOREST:**

**Problem Statement & Algorithm:**

****

**Solution:**

**Diabetes Dataset:**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import accuracy\_score**

**# Load the dataset**

**dataset\_path = '/content/diabetes.csv'**

**df = pd.read\_csv(dataset\_path)**

**# Split the dataset into features (X) and target (y)**

**X = df.drop(columns='Outcome') # Assuming 'Outcome' is the target column**

**y = df['Outcome']**

**# Split the data into training and test sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Initialize the Random Forest Classifier**

**rf\_model = RandomForestClassifier(random\_state=42)**

**# Train the model on the original features**

**rf\_model.fit(X\_train, y\_train)**

**# Get feature importance scores**

**importances = rf\_model.feature\_importances\_**

**# Create a DataFrame for feature importances**

**feature\_importance\_df = pd.DataFrame({**

**'Feature': X.columns,**

**'Importance': importances**

**})**

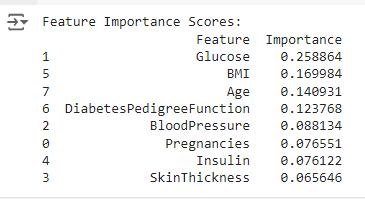
**# Sort the features by importance**

**feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False)**

**# Display the feature importance**

**print("Feature Importance Scores:")**

**print(feature\_importance\_df)**

****

**#Select the top 5 most important features**

**top\_5\_features = feature\_importance\_df.head(5)['Feature'].tolist()**

**print(f'\nTop 5 Most Important Features: {top\_5\_features}')**

****

**# Create reduced datasets with only the top 5 features**

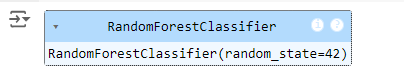
**X\_train\_reduced = X\_train[top\_5\_features]**

**X\_test\_reduced = X\_test[top\_5\_features]**

**# Train the model with the reduced feature set**

**rf\_model\_reduced = RandomForestClassifier(random\_state=42)**

**rf\_model\_reduced.fit(X\_train\_reduced, y\_train)**

****

**# Predict on the test set**

**y\_pred\_reduced = rf\_model\_reduced.predict(X\_test\_reduced)**

**# Calculate the accuracy with the reduced feature set**

**accuracy\_reduced = accuracy\_score(y\_test, y\_pred\_reduced)**

**print(f'Accuracy with top 5 features: {accuracy\_reduced:.4f}')**

****

**Melbourne Dataset:**

**import pandas as pd**

**import numpy as np**

**from sklearn.ensemble import RandomForestRegressor**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import mean\_absolute\_error**

**import matplotlib.pyplot as plt**

**# Load the dataset**

**data = pd.read\_csv('/content/melbourne\_housing\_raw.csv')**

**# Clean the dataset (handling NaN values, if necessary)**

**data.dropna(subset=['Price'], inplace=True) # Ensure we have prices**

**# Keep only numeric features for simplicity**

**numeric\_features = data.select\_dtypes(include=[np.number]).drop(columns=['Price']).columns.tolist()**

**X = data[numeric\_features]**

**y = data['Price']**

**# Split the dataset 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)**

**# Train a Random Forest model**

**model = RandomForestRegressor(random\_state=42)**

**model.fit(X\_train, y\_train)**

**# Predict on the test set and calculate the initial accuracy**

**y\_pred = model.predict(X\_test)**

**initial\_mae = mean\_absolute\_error(y\_test, y\_pred)**

**print(f"Initial MAE with all features: {initial\_mae:.2f}")**

**# Get feature importances**

**importances = model.feature\_importances\_**

**feature\_importances = pd.DataFrame({'Feature': X\_train.columns, 'Importance': importances})**

**feature\_importances = feature\_importances.sort\_values(by='Importance', ascending=False)**

**# Plot feature importances**

**plt.figure(figsize=(10, 6))**

**plt.barh(feature\_importances['Feature'], feature\_importances['Importance'])**

**plt.title('Feature Importance - Random Forest')**

**plt.xlabel('Importance')**

**plt.ylabel('Feature')**

**plt.gca().invert\_yaxis() # To display the most important feature on top**

**plt.show()**

**# Remove the least important features iteratively and evaluate performance**

**mae\_values = [initial\_mae] # To store MAE values**

**remaining\_features = X\_train.columns.tolist()**

**while len(remaining\_features) > 1:**

**# Remove the least important feature**

**least\_important\_feature = feature\_importances.iloc[-1]['Feature']**

**remaining\_features.remove(least\_important\_feature)**

**# Retrain the model with the remaining features**

**model.fit(X\_train[remaining\_features], y\_train)**

**y\_pred = model.predict(X\_test[remaining\_features])**

**# Calculate and store MAE**

**mae = mean\_absolute\_error(y\_test, y\_pred)**

**mae\_values.append(mae)**

**# Update feature importance table**

**feature\_importances = feature\_importances[feature\_importances['Feature'] != least\_important\_feature]**

**# Plot the MAE vs number of features**

**plt.figure(figsize=(10, 6))**

**plt.plot(range(len(mae\_values)), mae\_values, marker='o')**

**plt.title('MAE vs. Number of Features')**

**plt.xlabel('Number of Features Remaining')**

**plt.ylabel('Mean Absolute Error')**

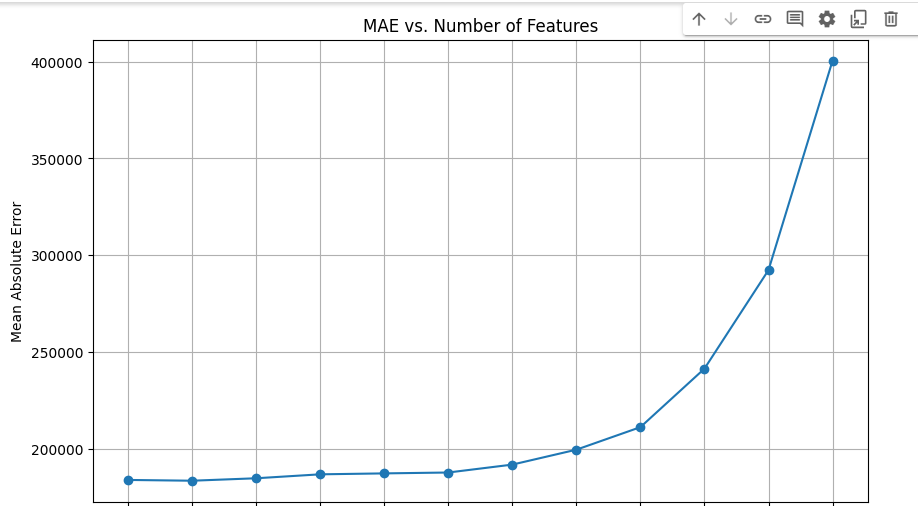
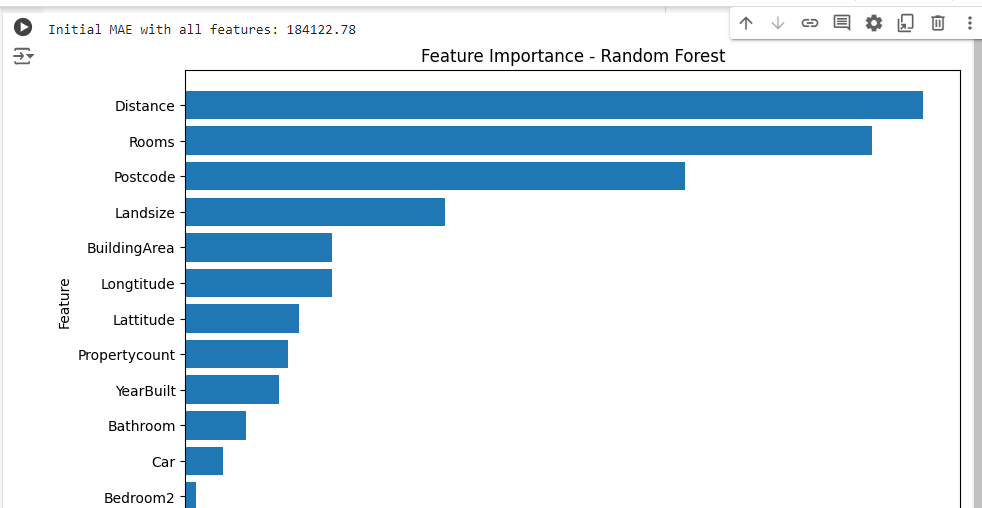
**plt.xticks(range(len(mae\_values)), range(len(X\_train.columns), 0, -1))**

**plt.grid()**

**plt.show()**

**# Final output: MAE with the last set of features**

**print(f"Final MAE with {len(remaining\_features)} features: {mae\_values[-1]:.2f}")**

****

****

**CODE REPOSITORY :**

**https://github.com/Roda1458/Feature-selection**