**FEATURE SELECTION**

**CSE 303: Machine Learning**

Submitted by

Name: Ridhi Guntur

Roll No: AP22110011467

Section: CSE M

Lab Date: 08/10/2024

Submission Date: 20/10/2024

**A picture containing text

Description automatically generated**

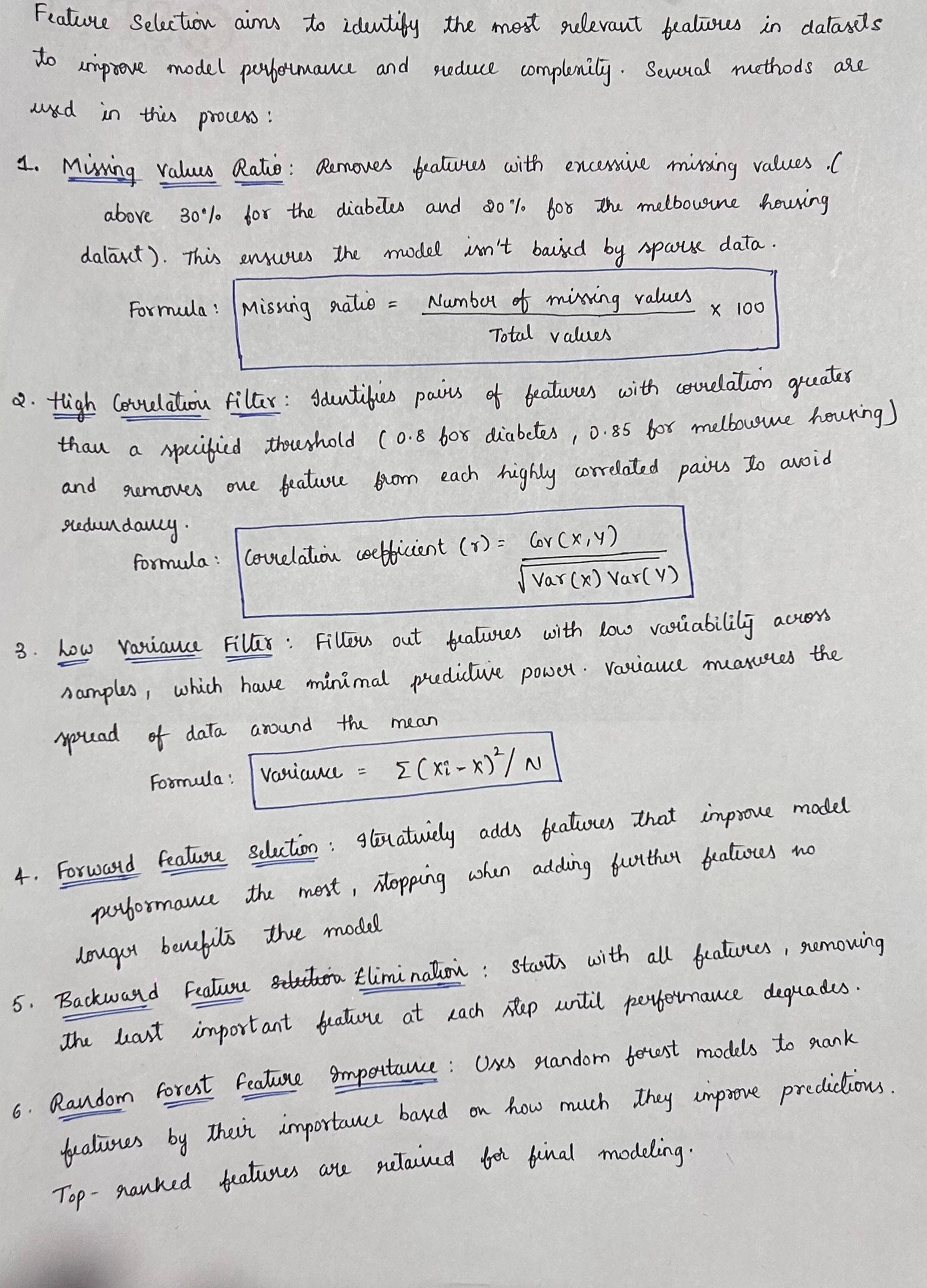
**Department Computer Science and Engineering**

**School of Engineering and Sciences**

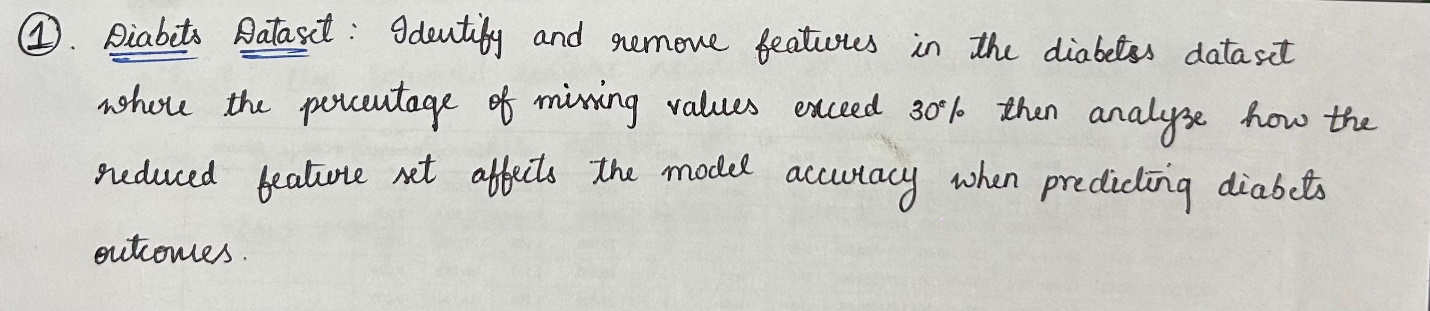
**SRM University–AP**

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

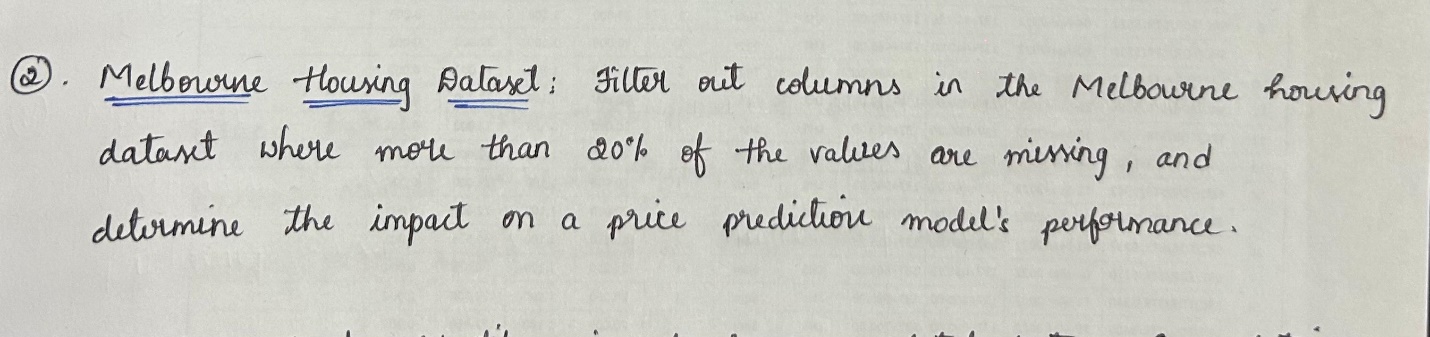
1. **Algorithm Description**



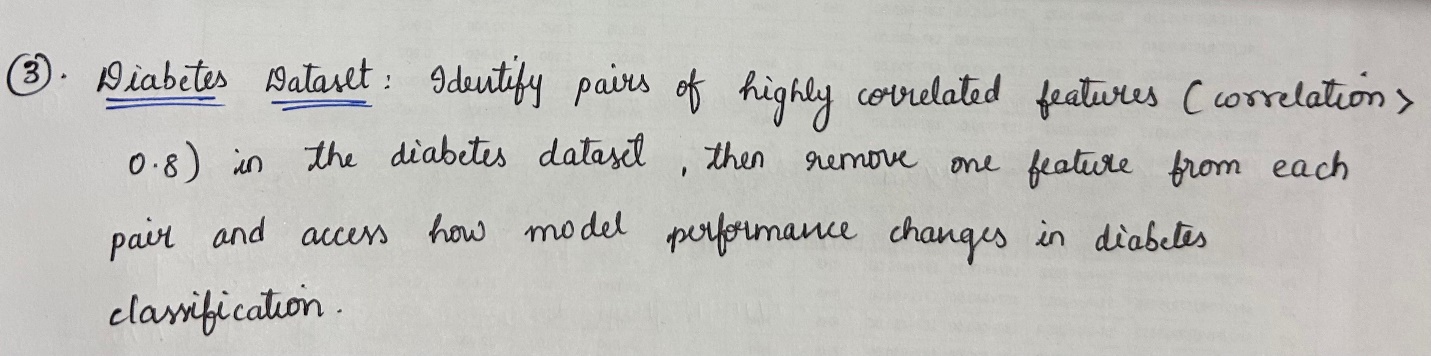
1. **Solutions**



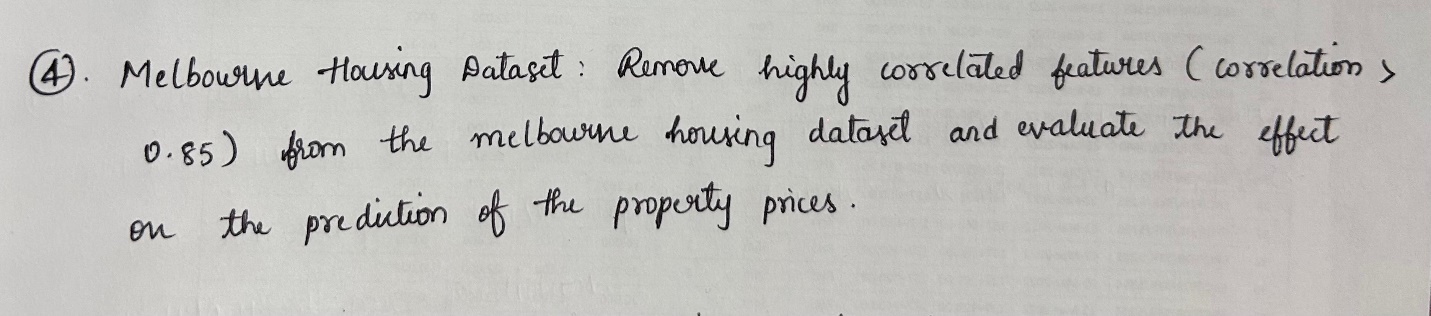
|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.ensemble import RandomForestClassifier  from sklearn.metrics import accuracy\_score  diabetes\_data = pd.read\_csv((r"C:\Users\guntu\Downloads\diabetes.csv"))  # Step 1: Count the number of zero values in relevant columns (Glucose, BloodPressure, etc.)  invalid\_columns = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']  zero\_counts = (diabetes\_data[invalid\_columns] == 0).sum()  # Step 2: Calculate the percentage of zero values in these columns  missing\_percentage = (zero\_counts / len(diabetes\_data)) \* 100  # Display the number and percentage of zero values  print("Number of zero values:")  print(zero\_counts)  print("\nPercentage of zero values:")  print(missing\_percentage)  # Step 3: Model accuracy before removing any columns  X\_full = diabetes\_data.drop(columns=['Outcome'])  y\_full = diabetes\_data['Outcome']  X\_train\_full, X\_test\_full, y\_train\_full, y\_test\_full = train\_test\_split(X\_full, y\_full, test\_size=0.2, random\_state=42)  # Train the RandomForest model on full data  model\_full = RandomForestClassifier(random\_state=42)  model\_full.fit(X\_train\_full, y\_train\_full)  y\_pred\_full = model\_full.predict(X\_test\_full)  accuracy\_full = accuracy\_score(y\_test\_full, y\_pred\_full)  print("\nModel Accuracy before removing columns: {:.2f}%".format(accuracy\_full \* 100))  columns\_to\_remove = missing\_percentage[missing\_percentage > 30].index  reduced\_data = diabetes\_data.drop(columns=columns\_to\_remove)  print("\nReduced dataset columns:")  print(reduced\_data.columns)  # Step 5: Model accuracy after removing columns with more than 30% missing values  X\_reduced = reduced\_data.drop(columns=['Outcome']) # Features after reduction  y\_reduced = reduced\_data['Outcome']  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)  model\_reduced = RandomForestClassifier(random\_state=42)  model\_reduced.fit(X\_train\_reduced, y\_train\_reduced)  y\_pred\_reduced = model\_reduced.predict(X\_test\_reduced)  accuracy\_reduced = accuracy\_score(y\_test\_reduced, y\_pred\_reduced)  print("\nModel Accuracy after removing columns: {:.2f}%".format(accuracy\_reduced \* 100)) |



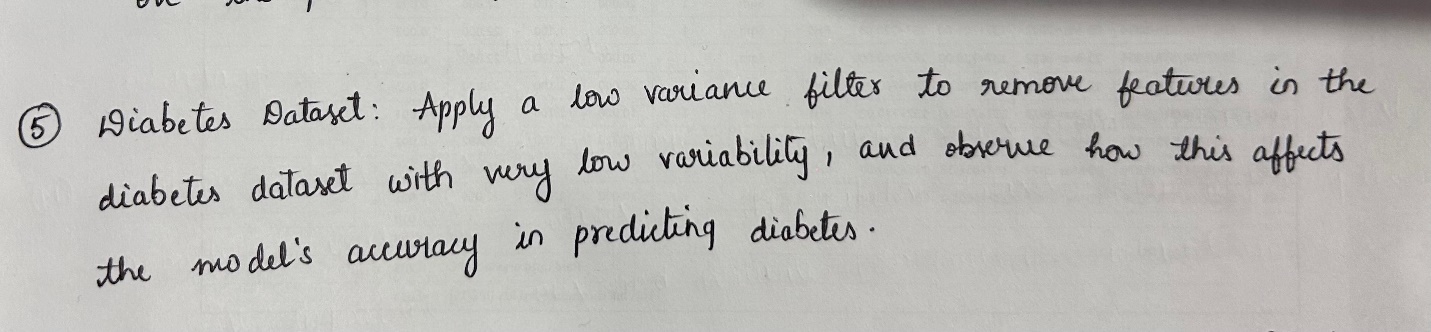
|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score  from sklearn.impute import SimpleImputer  from sklearn.preprocessing import StandardScaler  melbourne\_data = pd.read\_csv(r"C:\Users\guntu\Downloads\melbourne\_housing\_raw.csv")  missing\_counts = melbourne\_data.isnull().sum() # Count of missing values  missing\_ratios = melbourne\_data.isnull().mean() \* 100 # Percentage of missing values  print("Missing Values:")  for col in melbourne\_data.columns:  print(f"{col}: {missing\_counts[col]} ")  print("\nPercentage of Missing Values:")  for col in melbourne\_data.columns:  print(f"{col}: {missing\_ratios[col]:.2f}%")  columns\_to\_drop = missing\_ratios[missing\_ratios > 20].index.tolist()  print("\nColumns to be removed (more than 20% missing values):")  for col in columns\_to\_drop:  print(col)  if 'Price' in columns\_to\_drop:  columns\_to\_drop.remove('Price')  filtered\_data = melbourne\_data.drop(columns=columns\_to\_drop).dropna(subset=['Price'])  # Define features (X) and target (y)  X = filtered\_data.drop(columns=['Price'])  y = filtered\_data['Price']  X\_encoded = pd.get\_dummies(X, drop\_first=True)  imputer = SimpleImputer(strategy='median')  X\_imputed = imputer.fit\_transform(X\_encoded)  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X\_imputed)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  model = LinearRegression()  model.fit(X\_train, y\_train)  y\_pred = model.predict(X\_test)  mae = mean\_absolute\_error(y\_test, y\_pred)  mse = mean\_squared\_error(y\_test, y\_pred)  r2 = r2\_score(y\_test, y\_pred)  print(f'\nMean Absolute Error (MAE): {mae:.2f}')  print(f'Mean Squared Error (MSE): {mse:.2f}')  print(f'R² Score: {r2:.2f}') |



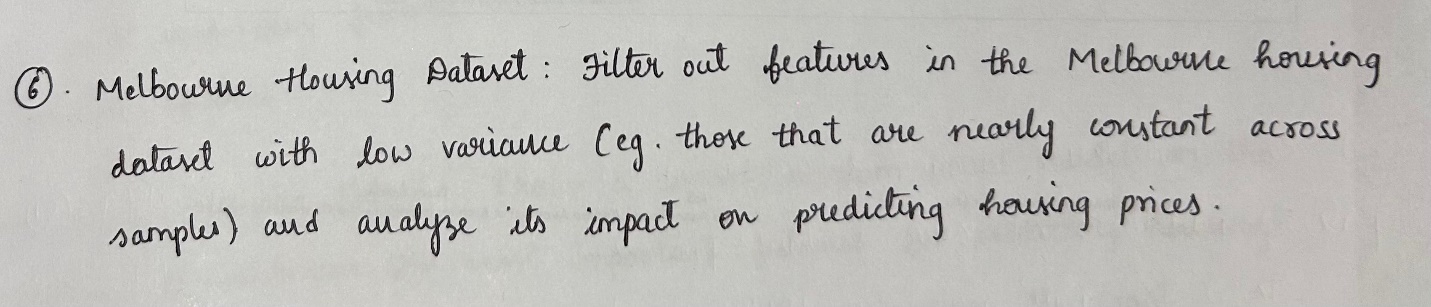
|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import accuracy\_score  data = pd.read\_csv((r"C:\Users\guntu\Downloads\diabetes.csv"))  # Compute the correlation matrix  print("Computing the correlation matrix \n")  corr\_matrix = data.corr()  print(corr\_matrix, "\n")  corr\_pairs = []  for i in range(len(corr\_matrix.columns)):  for j in range(i + 1, len(corr\_matrix.columns)):  corr\_value = corr\_matrix.iloc[i, j]  if abs(corr\_value) > 0.8:  corr\_pairs.append((corr\_matrix.columns[i], corr\_matrix.columns[j], corr\_value))  if corr\_pairs:  print("Highly Correlated Pairs:")  for pair in corr\_pairs:  print(f"{pair[0]} and {pair[1]}: Correlation = {pair[2]:.2f}")  else:  print("No pairs with correlation > 0.8 found.\n")  features\_to\_remove = set(pair[1] for pair in corr\_pairs) # Keep only one feature from each pair  data\_reduced = data.drop(columns=features\_to\_remove)  print(f"Features removed: {features\_to\_remove}\n")  # Split the data into train and test sets  X = data.drop(columns='Outcome')  X\_reduced = data\_reduced.drop(columns='Outcome')  y = data['Outcome']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  X\_train\_reduced, X\_test\_reduced, \_, \_ = train\_test\_split(X\_reduced, y, test\_size=0.3, random\_state=42)  # Train models on both versions of the dataset  model = LogisticRegression(max\_iter=500)  model\_reduced = LogisticRegression(max\_iter=500)  model.fit(X\_train, y\_train)  model\_reduced.fit(X\_train\_reduced, y\_train)  # Evaluate model performance  y\_pred = model.predict(X\_test)  y\_pred\_reduced = model\_reduced.predict(X\_test\_reduced)  accuracy = accuracy\_score(y\_test, y\_pred)  print(f"Accuracy with all features: {accuracy:.4f}")  accuracy\_reduced = accuracy\_score(y\_test, y\_pred\_reduced)  print(f"Accuracy with reduced features: {accuracy\_reduced:.4f}\n") |



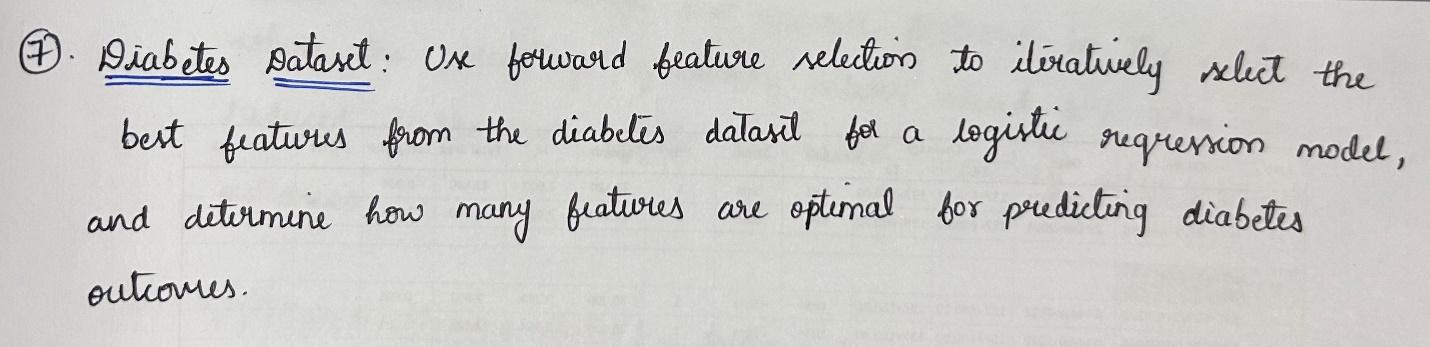
|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error, r2\_score  from sklearn.preprocessing import OneHotEncoder  import seaborn as sns  import matplotlib.pyplot as plt  # Load the dataset  df = pd.read\_csv(r"C:\Users\guntu\Downloads\melbourne\_housing\_raw.csv")  # Handle missing values by dropping rows with missing values  df\_cleaned = df.dropna()  # Verify there are no missing values  print(df\_cleaned.isnull().sum())  # Remove non-numeric columns before calculating the correlation matrix  df\_numeric = df\_cleaned.select\_dtypes(include=[np.number])  # Compute the correlation matrix for numeric columns only  correlation\_matrix = df\_numeric.corr()  print(correlation\_matrix)  # Identify and remove highly correlated features (correlation > 0.85)  threshold = 0.85  high\_corr\_features = np.where(correlation\_matrix > threshold)  high\_corr\_pairs = [(correlation\_matrix.index[x], correlation\_matrix.columns[y]) for x, y in zip(\*high\_corr\_features) if x != y and x < y]  print("Highly correlated features (correlation > 0.85): \n " , high\_corr\_pairs)  # Remove one feature from each highly correlated pair  features\_to\_remove = set([pair[1] for pair in high\_corr\_pairs])  df\_reduced = df\_cleaned.drop(columns=features\_to\_remove)  # Convert categorical variables into numerical values using OneHotEncoder  df\_reduced = pd.get\_dummies(df\_reduced, drop\_first=True)  X = df\_reduced.drop(columns=['Price'])  y = df\_reduced['Price']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Train a linear regression model  model = LinearRegression()  model.fit(X\_train, y\_train)  y\_pred = model.predict(X\_test)  mse = mean\_squared\_error(y\_test, y\_pred)  print(f"Mean Squared Error: {mse}")  r2 = r2\_score(y\_test, y\_pred)  print(f"R-squared: {r2}") |



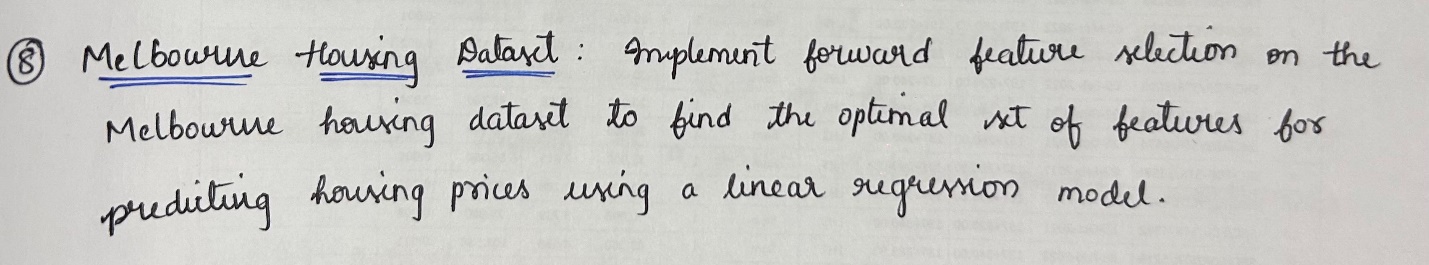
|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import accuracy\_score  from sklearn.feature\_selection import VarianceThreshold  data = pd.read\_csv(r"C:\Users\guntu\Downloads\diabetes.csv")  X = data.drop(columns='Outcome')  y = data['Outcome']  # Calculate the variance of each feature  variances = X.var()  print("Variance of features:\n", variances, "\n")  # Apply a low variance filter with a threshold of 1.0  selector = VarianceThreshold(threshold=1.0)  X\_reduced = selector.fit\_transform(X)  # Get the names of features that remain after filtering  remaining\_features = X.columns[selector.get\_support()]  print(f"Remaining Features after Low Variance Filter: {list(remaining\_features)}")  print(f"Reduced dataset shape: {X\_reduced.shape}\n")  # Split data into train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  X\_train\_reduced, X\_test\_reduced, \_, \_ = train\_test\_split(X\_reduced, y, test\_size=0.3, random\_state=42)  # Train Logistic Regression models  model = LogisticRegression(max\_iter=500)  model\_reduced = LogisticRegression(max\_iter=500)  model.fit(X\_train, y\_train)  model\_reduced.fit(X\_train\_reduced, y\_train)  y\_pred = model.predict(X\_test)  y\_pred\_reduced = model\_reduced.predict(X\_test\_reduced)  accuracy = accuracy\_score(y\_test, y\_pred)  accuracy\_reduced = accuracy\_score(y\_test, y\_pred\_reduced)  print(f"Accuracy with all features: {accuracy:.4f}")  print(f"Accuracy with reduced features: {accuracy\_reduced:.4f}\n") |



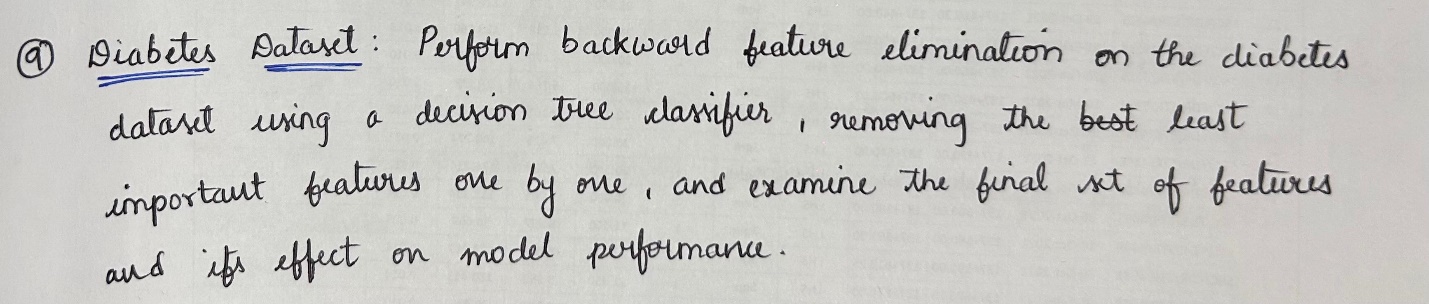
|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.feature\_selection import VarianceThreshold  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error, r2\_score  # Load the dataset from the specified file path  df = pd.read\_csv(r"C:\Users\guntu\Downloads\melbourne\_housing\_raw.csv")  # Handle missing values by dropping rows with missing values  df\_cleaned = df.dropna()  # Convert categorical variables into numerical values using OneHotEncoder or get\_dummies  df\_encoded = pd.get\_dummies(df\_cleaned, drop\_first=True)  X = df\_encoded.drop(columns=['Price'])  y = df\_encoded['Price']  # Calculate variance of all features  feature\_variances = X.var()  print("Variance of features before filtering:")  print(feature\_variances)  # Apply Variance Threshold (Filter out features with variance below the threshold, e.g., 0.01)  variance\_threshold = VarianceThreshold(threshold=0.01)  X\_high\_variance = variance\_threshold.fit\_transform(X)  # Get the indices of the features that were kept and removed  features\_kept = X.columns[variance\_threshold.get\_support()]  features\_removed = X.columns[~variance\_threshold.get\_support()]  print("\nFeatures removed after applying variance threshold:")  print(features\_removed)  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)  y\_pred = model.predict(X\_test)  mse = mean\_squared\_error(y\_test, y\_pred)  r2 = r2\_score(y\_test, y\_pred)  print(f"\nMean Squared Error after removing low variance features: {mse}")  print(f"R-squared after removing low variance features: {r2}")  # Compare with model performance before filtering low variance  X\_train\_full, X\_test\_full, y\_train\_full, y\_test\_full = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  model\_full = LinearRegression()  model\_full.fit(X\_train\_full, y\_train\_full)  y\_pred\_full = model\_full.predict(X\_test\_full)  mse\_full = mean\_squared\_error(y\_test\_full, y\_pred\_full)  r2\_full = r2\_score(y\_test\_full, y\_pred\_full)  print(f"\nMean Squared Error before removing low variance features: {mse\_full}")  print(f"R-squared before removing low variance features: {r2\_full}") |



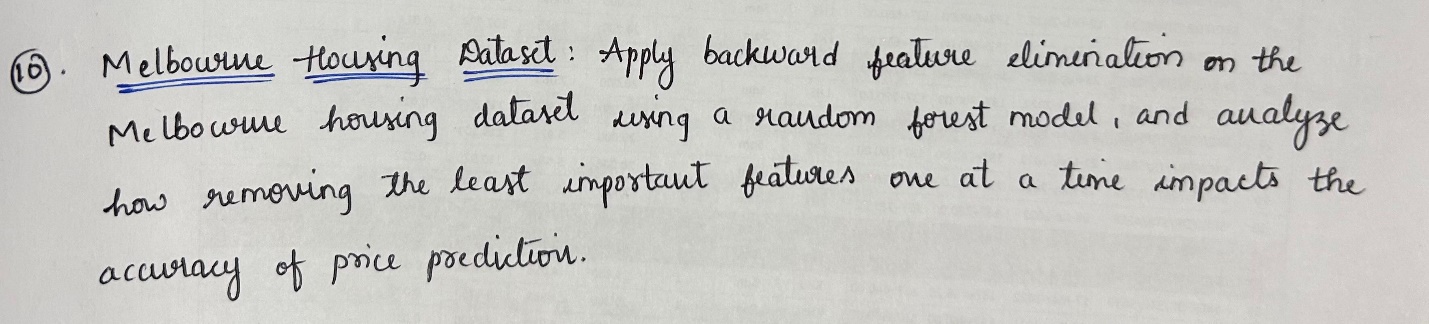
|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split, cross\_val\_score  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import accuracy\_score  from sklearn.feature\_selection import SequentialFeatureSelector  data = pd.read\_csv(r"C:\Users\guntu\Downloads\diabetes.csv")  X = data.drop(columns='Outcome')  y = data['Outcome']  # Initialize Logistic Regression model  logreg = LogisticRegression(max\_iter=500)  # Apply Forward Feature Selection  # Using SequentialFeatureSelector for forward selection  sfs = SequentialFeatureSelector(  logreg, direction='forward', scoring='accuracy', cv=5  )  sfs.fit(X, y)  # Get the selected features  selected\_features = X.columns[sfs.get\_support()]  print(f"Selected Features: {list(selected\_features)}")  print(f"Number of selected features: {len(selected\_features)}\n")  # Train model with selected features  X\_selected = X[selected\_features]  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_selected, y, test\_size=0.3, random\_state=42)  logreg.fit(X\_train, y\_train)  # Evaluate the model  y\_pred = logreg.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  print(f"Accuracy with selected features: {accuracy:.4f}\n") |



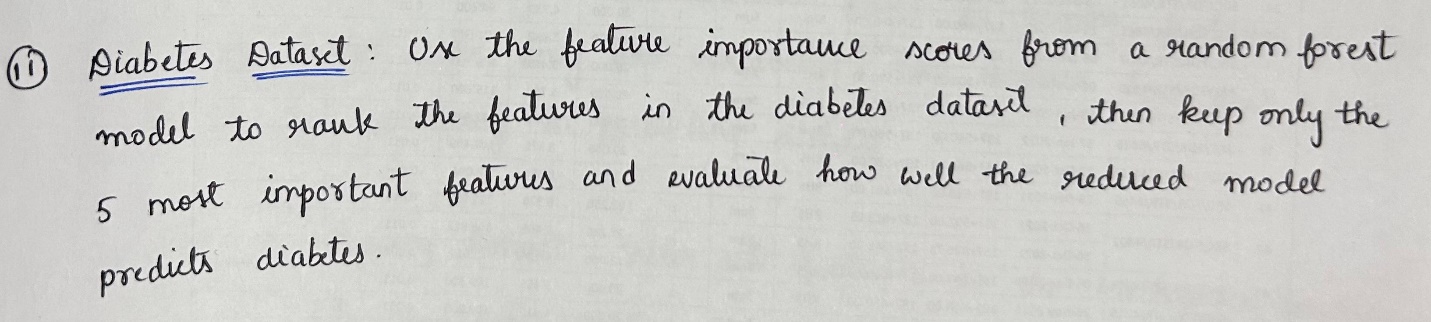
|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split, cross\_val\_score  from sklearn.linear\_model import LinearRegression  from sklearn.feature\_selection import SequentialFeatureSelector  from sklearn.metrics import mean\_absolute\_error, make\_scorer  from sklearn.impute import SimpleImputer  melbourne\_data = pd.read\_csv(r"C:\Users\guntu\Downloads\melbourne\_housing\_raw.csv")  data\_clean = melbourne\_data.dropna(subset=['Price'])  X = data\_clean.drop(columns=['Price'])  y = data\_clean['Price']  numerical\_features = X.select\_dtypes(include=['float64', 'int64'])  imputer = SimpleImputer(strategy='mean')  X\_imputed = pd.DataFrame(imputer.fit\_transform(numerical\_features), columns=numerical\_features.columns)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_imputed, y, test\_size=0.2, random\_state=42)  model = LinearRegression()  mae\_scorer = make\_scorer(mean\_absolute\_error, greater\_is\_better=False)  selector = SequentialFeatureSelector(  model,  n\_features\_to\_select="auto",  direction='forward',  scoring=mae\_scorer,  cv=5  )  selector.fit(X\_train, y\_train)  selected\_features = X\_imputed.columns[selector.get\_support()]  print("Selected Features:")  print(selected\_features)  X\_train\_selected = selector.transform(X\_train)  X\_test\_selected = selector.transform(X\_test)  model.fit(X\_train\_selected, y\_train)  y\_pred = model.predict(X\_test\_selected)  mae = mean\_absolute\_error(y\_test, y\_pred)  print(f"\nMean Absolute Error (MAE) with selected features: {mae:.2f}") |



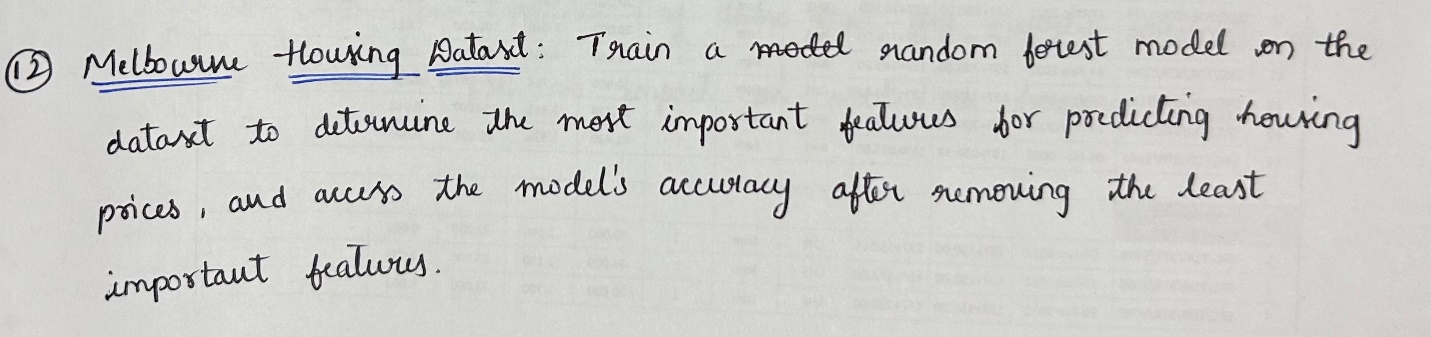
|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score  from sklearn.feature\_selection import RFE  data = pd.read\_csv(r"C:\Users\guntu\Downloads\diabetes.csv")  X = data.drop(columns='Outcome')  y = data['Outcome']  # Initialize Decision Tree Classifier  model = DecisionTreeClassifier(random\_state=42)  # Apply Recursive Feature Elimination (RFE) for backward feature elimination  rfe = RFE(estimator=model, n\_features\_to\_select=1) # Keep reducing until 1 feature remains  rfe.fit(X, y)  # Print ranking of features (1 = most important)  for feature, rank in zip(X.columns, rfe.ranking\_):  print(f"{feature}: Rank {rank}")  # Select the optimal set of features (those with rank 1)  selected\_features = X.columns[rfe.support\_]  print(f"\nFinal Selected Features: {list(selected\_features)}")  print(f"Number of Selected Features: {len(selected\_features)}\n")  # Train the model with selected features  X\_selected = X[selected\_features]  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_selected, y, test\_size=0.3, random\_state=42)  model.fit(X\_train, y\_train)  #Evaluate the model performance  y\_pred = model.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  print(f"Accuracy with selected features: {accuracy:.4f}\n") |



|  |
| --- |
| import pandas as pd  from sklearn.ensemble import RandomForestRegressor  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_absolute\_error  from sklearn.feature\_selection import RFE  from sklearn.impute import SimpleImputer  melbourne\_data = pd.read\_csv(r"C:\Users\guntu\Downloads\melbourne\_housing\_raw.csv")  melbourne\_data\_cleaned = melbourne\_data.dropna(subset=['Price'])  X = melbourne\_data\_cleaned.select\_dtypes(include=['float64', 'int64']).drop(columns=['Price'])  y = melbourne\_data\_cleaned['Price']  imputer = SimpleImputer(strategy='mean')  X\_imputed = imputer.fit\_transform(X)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_imputed, y, test\_size=0.2, random\_state=42)  model = RandomForestRegressor(random\_state=42)  rfe = RFE(estimator=model, n\_features\_to\_select=1, step=1)  rfe.fit(X\_train, y\_train)  ranking = rfe.ranking\_  feature\_names = melbourne\_data\_cleaned.select\_dtypes(include=['float64', 'int64']).drop(columns=['Price']).columns  feature\_ranking = pd.DataFrame({'Feature': feature\_names, 'Ranking': ranking})  feature\_ranking\_sorted = feature\_ranking.sort\_values(by='Ranking')  print("Feature Rankings:")  print(feature\_ranking\_sorted)  mae\_list = []  features\_left = list(feature\_names)  for i in range(1, len(features\_left)):  # Select top i features (excluding least important features)  selected\_features = feature\_ranking\_sorted['Feature'].head(len(features\_left) - i)  X\_train\_selected = pd.DataFrame(X\_train, columns=feature\_names)[selected\_features]  X\_test\_selected = pd.DataFrame(X\_test, columns=feature\_names)[selected\_features]  # Train and evaluate model with selected features  model.fit(X\_train\_selected, y\_train)  y\_pred = model.predict(X\_test\_selected)  mae = mean\_absolute\_error(y\_test, y\_pred)  mae\_list.append((len(selected\_features), mae))  print("\n Performance after feature elimination:")  for num\_features, mae in mae\_list:  print(f'Total features : {num\_features}, MAE: {mae}') |



|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.ensemble import RandomForestClassifier  from sklearn.metrics import accuracy\_score  data = pd.read\_csv(r"C:\Users\guntu\Downloads\diabetes.csv")  X = data.drop(columns='Outcome')  y = data['Outcome']  # Initialize Random Forest Classifier  rf\_model = RandomForestClassifier(random\_state=42)  # Fit the model  rf\_model.fit(X, y)  # Get feature importance scores  importance\_scores = rf\_model.feature\_importances\_  feature\_importance = pd.Series(importance\_scores, index=X.columns).sort\_values(ascending=False)  # Print feature importance scores  print("Feature Importance Scores:")  print(feature\_importance, "\n")  # Select the top 5 most important features  top\_features = feature\_importance.head(5).index.tolist()  print(f"Top 5 Features: {top\_features}\n")  X\_top = X[top\_features]  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_top, y, test\_size=0.3, random\_state=42)  rf\_model.fit(X\_train, y\_train)  # Evaluate the model performance  y\_pred = rf\_model.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  print(f"Accuracy with top 5 features: {accuracy:.4f}\n") |



|  |
| --- |
| import pandas as pd  from sklearn.ensemble import RandomForestRegressor  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_absolute\_error  from sklearn.impute import SimpleImputer  import numpy as np  melbourne\_data = pd.read\_csv(r"C:\Users\guntu\Downloads\melbourne\_housing\_raw.csv")  melbourne\_data\_cleaned = melbourne\_data.dropna(subset=['Price'])  X = melbourne\_data\_cleaned.select\_dtypes(include=['float64', 'int64']).drop(columns=['Price'])  y = melbourne\_data\_cleaned['Price']  imputer = SimpleImputer(strategy='mean')  X\_imputed = imputer.fit\_transform(X)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_imputed, y, test\_size=0.2, random\_state=42)  model = RandomForestRegressor(random\_state=42)  model.fit(X\_train, y\_train)  importances = model.feature\_importances\_  indices = np.argsort(importances)[::-1] # Sort by importance descending  feature\_names = melbourne\_data\_cleaned.select\_dtypes(include=['float64', 'int64']).drop(columns=['Price']).columns  print("Feature Importance Ranking:")  for idx in indices:  print(f'{feature\_names[idx]}: {importances[idx]}')  y\_pred = model.predict(X\_test)  mae\_full = mean\_absolute\_error(y\_test, y\_pred)  print(f'\nModel accuracy with all features, MAE: {mae\_full}')  n\_least\_important = 3  least\_important\_features = feature\_names[indices[-n\_least\_important:]]  selected\_features\_idx = indices[:-n\_least\_important]  X\_train\_reduced = X\_train[:, selected\_features\_idx]  X\_test\_reduced = X\_test[:, selected\_features\_idx]  model.fit(X\_train\_reduced, y\_train)  y\_pred\_reduced = model.predict(X\_test\_reduced)  mae\_reduced = mean\_absolute\_error(y\_test, y\_pred\_reduced)  print(f'\nRemoved the {n\_least\_important} least important features: {list(least\_important\_features)}')  print(f'Model accuracy after removing these features, MAE: {mae\_reduced}') |

1. **Link**