

Start coding or [upload](#) with AI.

Double-click (or enter) to edit

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

The Final project is a study of diabetic patients Dataset.

Dr:Hager Saleh

Student:Jihad Abdullah Alharibi,253002789

Dataset description

```
import warnings
warnings.filterwarnings("ignore")
# =====
# Core libraries
# =====
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# =====
# Load data - data handling
# =====
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline

# =====
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
# =====
# Cross-validation metrics
# =====
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)
# =====
# Imbalanced learn: sampling methods
# =====
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek

df = pd.read_csv('content/diabetes.csv')

df.dtypes

#
# Pregnancies int64
# Glucose int64
# BloodPressure int64
# SkinThickness int64
# Insulin int64
# BMI float64
# DiabetesPedigreeFunction float64
# Age int64
# Outcome int64

dfpe:object

df.head()

#
# Pregnancies 0
# Glucose 148
# BloodPressure 70
# SkinThickness 35
# Insulin 0
# BMI 23.0
# DiabetesPedigreeFunction 0.357
# Age 36
# Outcome 1

#
# Pregnancies 1
# Glucose 85
# BloodPressure 66
# SkinThickness 66
# Insulin 0
# BMI 26.6
# DiabetesPedigreeFunction 0.361
# Age 31
# Outcome 0

#
# Pregnancies 8
# Glucose 183
# BloodPressure 64
# SkinThickness 0
# Insulin 0
# BMI 23.3
# DiabetesPedigreeFunction 0.672
# Age 32
# Outcome 1

#
# Pregnancies 1
# Glucose 89
# BloodPressure 66
# SkinThickness 23
# Insulin 94
# BMI 28.1
# DiabetesPedigreeFunction 0.167
# Age 21
# Outcome 0

#
# Pregnancies 0
# Glucose 137
# BloodPressure 40
# SkinThickness 35
# Insulin 168
# BMI 43.1
# DiabetesPedigreeFunction 2.288
# Age 33
# Outcome 1

#
# Pregnancies 0
# Glucose 0
# BloodPressure 0
# SkinThickness 0
# Insulin 0
# BMI 0
# DiabetesPedigreeFunction 0
# Age 0
# Outcome 0

dfpe:ndarray

df['Glucose'].value_counts()

#
# count
# Glucose
# 99 17
# 100 17
# 101 14
# 125 14
# 129 14
# -- --
# 96 1
# 103 1
# 140 1
# 85 1
# 100 1
# 125 nan 1 outcome

dfpe:ndarray

df.isnull().sum()

#
# Pregnancies 0
# Glucose 0
# BloodPressure 0
# SkinThickness 0
# Insulin 0
# BMI 0
# DiabetesPedigreeFunction 0
# Age 0
# Outcome 0

dfpe:ndarray

df.drop(['Pregnancies'], axis=1, inplace=True)

df['BMI'].fillna('Unknown', inplace=True)

df.isnull().sum()

#
# Glucose 0
# BloodPressure 0
# SkinThickness 0
# Insulin 0
# BMI 0
# DiabetesPedigreeFunction 0
# Age 0
# Outcome 0

dfpe:ndarray

df.describe()

#
# count 768.000000
# mean 33.066324
# std 31.972618
# min 0.000000
# 25% 96.000000
# 50% 117.000000
# 75% 140.260000
# max 199.000000

#
# BloodPressure 78.000000
# mean 70.706476
# std 19.358607
# min 0.000000
# 25% 96.000000
# 50% 72.000000
# 75% 80.000000
# max 122.000000

#
# SkinThickness 768.000000
# mean 70.706476
# std 15.952218
# min 0.000000
# 25% 62.000000
# 50% 72.000000
# 75% 80.000000
# max 99.000000

#
# Insulin 768.000000
# mean 70.706476
# std 115.244802
# min 0.000000
# 25% 0.000000
# 50% 23.000000
# 75% 127.000000
# max 846.000000

#
# BMI 768.000000
# mean 26.745678
# std 7.884160
# min 0.000000
# 25% 24.000000
# 50% 26.500000
# 75% 34.000000
# max 67.000000

#
# DiabetesPedigreeFunction 768.000000
# mean 0.471678
# std 0.331329
# min 0.076000
# 25% 0.243750
# 50% 0.372500
# 75% 0.626250
# max 2.420000

#
# Age 768.000000
# mean 33.066324
# std 11.952470
# min 21.000000
# 25% 24.000000
# 50% 29.000000
# 75% 31.000000
# max 81.000000

#
# Outcome 768.000000
# mean 0.348946
# std 0.476891
# min 0.000000
# 25% 0.000000
# 50% 0.000000
# 75% 1.000000
# max 1.000000

from sklearn.datasets import load_iris
import pandas as pd

Start coding or upload with AI.
```

df.columns

```
Diabetes[[ 'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
          'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome' ],
         dtype=object)
```

df.columns

```
Diabetes[[ 'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
          'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome' ],
         dtype=object)
```

df.isnull().sum()

```
#
# Pregnancies 0
# Glucose 0
# BloodPressure 0
# SkinThickness 0
# Insulin 0
# BMI 0
# DiabetesPedigreeFunction 0
# Age 0
# Outcome 0

dfpe:ndarray

missing_percentage = df.isnull().sum() * 100
threshold = 10

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

for column in df.select_dtypes(include='object').columns:
```

https://colab.research.google.com/drive/1KDRXAAta29SzvxNLfJPIBLHXd_r7mKft#scrollTo=U0MrbYRBP1Pk&printMode=true

1/4

```
df[column] = LabelEncoder().transform(df[column])
df.head()
```

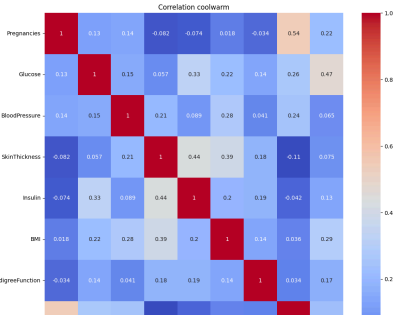
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.8	0.351	31	0
2	8	183	66	0	0	23.5	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.268	33	1

```
for column in df.select_dtypes(include='float64').columns:
    max_value = df[column].max() # Get the max (most frequent)
    df[column] = df[column].fillna(max_value)
# Fill missing values in categorical columns with mode
for column in df.select_dtypes(include='object').columns:
    mode_value = df[column].mode()[0] # Get the mode (most frequent)
    df[column] = df[column].fillna(mode_value)
df.isnull().sum()
```

```
plt.figure(figsize=(12, 12))
```

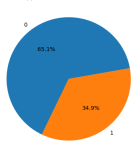
```
sm = heatmap(df_corr(), annot=True, cmap='coolwarm')
plt.title('Correlation coeefware')
```

```
text(0.5, 1.4, 'Correlation coeefware')
```



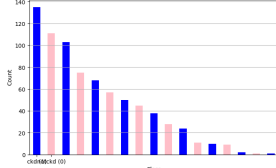
```
appet_counts = df['Outcome'].value_counts()
plt.figure(figsize=(5, 5))
plt.pie(appet_counts, labels=appet_counts.index, autopct='%1.1f%%',
        startangle=90)
plt.title('Appetite Status Distribution')
plt.show()
```

Appetite Status Distribution



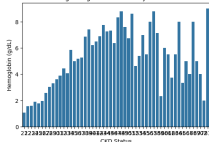
```
plt.figure(figsize=(8, 5))
df['Pregnancies'].value_counts().plot(kind='bar', color='blue',
        label='Pregnancies')
plt.xticks([0, 1], ['0 (65.1%)', '1 (34.9%)'], rotation=45)
plt.title('Pregnancies Kidney Disease Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```

Pregnancies Kidney Disease Distribution



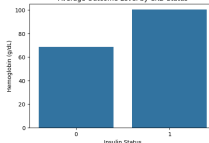
```
import seaborn as sns
plt.figure(figsize=(4, 4))
sns.barplot(x='Age', y='Pregnancies', data=df, estimator='mean',
        ci=None)
plt.title('Average Pregnancies Level by CKD Status')
plt.xlabel('CKD Status')
plt.ylabel('Hemoglobin (g/dL)')
plt.show()
```

Average Pregnancies Level by CKD Status



```
import seaborn as sns
plt.figure(figsize=(4, 4))
sns.barplot(x='Outcome', y='Insulin', data=df, estimator='mean',
        ci=None)
plt.title('Average Age Outcome Level by CKD Status')
plt.xlabel('Outcome Status')
plt.ylabel('Hemoglobin (g/dL)')
plt.show()
```

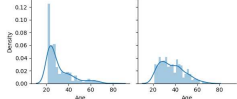
Average Outcome Level by CKD Status



```
g = sns.FacetGrid(df, col='Outcome')
```

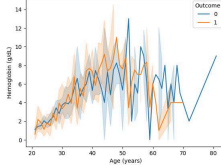
```
g.map(sns.distplot, 'Age', bins=20)
```

```
plt.show()
```

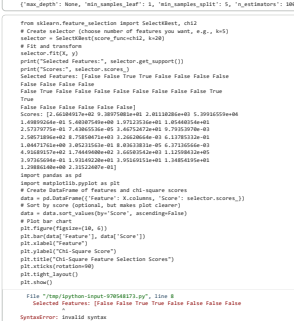


```
sns.lmplot(x='Age', y='Pregnancies', hue='Outcome', data=df)
plt.title('Hemoglobin vs Age by CKD Status')
plt.xlabel('Age (years)')
plt.ylabel('Hemoglobin (g/dL)')
plt.show()
```

Hemoglobin vs Age by CKD Status



```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Age', y='Glucose', data=df, hue='Pregnancies',
        palette='Set1')
plt.title('Age vs Hemoglobin')
plt.xlabel('Age')
plt.ylabel('Glucose')
plt.show()
```



```

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Get feature importances from the Best Random Forest Classifier
feature_importances = best_rf_classifier.feature_importances_

# Get feature names from the training data (X)
feature_names = X.columns

# Create a DataFrame for better visualization
importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': feature_importances
})

# Sort the DataFrame by importance in descending order
importance_df = importance_df.sort_values(by='importance', ascending=False)

# Print the feature importances
plt.figure(figsize=(10, 8))

sns.barplot(importance_df[['feature', 'importance']])

plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importance')
plt.show()

```

```

from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=42)
# Apply RFE (select top 5 features)
rfe = RFE(estimator=rf, n_features_to_select=5)
rfe.fit(X, y)
print("Selected Features Mask:", rfe.get_support())
print("Ranking of Features:", rfe.ranking_)

import pandas as pd
ranking_df = pd.DataFrame({
    "feature": X.columns,
    "RFE_Ranking": rfe.ranking_,
    "sort_order": RFE_Ranking
})

Selected Features Mask: [ True  True  True  True  True  True  True]
Ranking of Features: [1 2 1 1 1 2 1]

```

```

# Import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.bar(ranking_df['feature'], ranking_df['RFE_Ranking'])
plt.xlabel('Features')
plt.ylabel('RFE Ranking (1 = Selected)')
plt.title('Feature Ranking Using RFE + Random Forest')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```

```

# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)

# Extract Feature Importances
importance = rf.feature_importances_

# Use Feature names from X (current and safe)
feature_names = X.columns if hasattr(X, 'columns') else [f'Feature {i}' for i in range(X.shape[1])]

# Create DataFrame
df_importance = pd.DataFrame({
    'feature': feature_names,
    'importance': importance
})

# Sort by importance
df_importance = df_importance.sort_values(by='importance', ascending=False)

# Plot
plt.figure(figsize=(10, 6))
plt.bar(df_importance['feature'], df_importance['importance'])
plt.xlabel('Feature Importance')
plt.ylabel('Random Forest Feature Importance')
plt.tight_layout()
plt.show()

```

Feature	RFE_Ranking
Pregnancies	1
Glucose	1
BloodPressure	1
SkinThickness	1
Insulin	1
BMI	1
DiabetesPedigreeFunction	1
Age	1

Next steps: [Generate code with ranking API](#) [New interactive sheet](#)

```

df.importance().sum()

```

```

# Get Feature Importance scores
importance = model.feature_importances_

# Create DataFrame
data = pd.DataFrame({
    'feature': X.columns,
    'importance': importance
})

# Sort by importance
data = data.sort_values(by='importance', ascending=False)
print("Top Features Selected by Random Forest Model:")
print(data)

# Plot Feature Importance
plt.figure(figsize=(10, 6))
plt.bar(data['feature'], data['importance'])
plt.xlabel('Feature Importance')
plt.ylabel('Random Forest Feature Importance')
plt.tight_layout()
plt.show()

```

```

NameError                                Traceback (most recent call last)
<ipython-input-25625658-0> in <cell> 6()
.....: 1 # Get Feature Importance scores
.....: 2 importance = model.feature_importances_
.....: 3 # Create DataFrame
.....: 4 data = pd.DataFrame({
.....:     5 'feature': X.columns,
NameError: name 'model' is not defined

```

Next steps: [Copy error](#)

```

from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedShuffleSplit
import matplotlib.pyplot as plt

# Initialize your estimator (e.g., RandomForestClassifier)
estimator = RandomForestClassifier(random_state=42)

# Initialize RFE
rfe = RFE(estimator=estimator, step=1, cv_split=StratifiedShuffleSplit(2, scoring='accuracy', n_jobs=-1))

# Fit RFE to your data
rfe.fit(X, y)

print(f"Optimal number of Features: {rfe.n_features_}")
print(f"Selected Features: {list(X.columns[rfe.get_support()])}")

# Plot the number of features vs. cross-validation scores
plt.figure(figsize=(10, 6))
plt.plot(rfe.get_support(), rfe.score(X, y))
plt.xlabel('Number of Features Selected')
plt.ylabel('Cross validation score (accuracy)')
plt.grid(True)
plt.title(f'RFEV: Optimal Number of Features')
plt.show()

```

Optimal number of Features: 7
Selected Features: ['Pregnancies', 'Glucose', 'BloodPressure', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']