❖Project Source Code and Result

1 AI Diabetes Prediction using Python + Pandas:

Diabetes is a chronic (long-lasting) health condition that affects how your body turns food into energy. Most of the food you eat is broken down into sugar (also called glucose) and released into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin.

2 Table Content

3 Introduction

According to WHO, Diabetes is a chronic disease that occurs either when the pancreas does not produce enough insulin or when the body cannot effectively use the insulin it produces. Insulin is a hormone that regulates blood sugar. Hyperglycaemia, or raised blood sugar, is a common effect of uncontrolled diabetes and over time leads to serious damage to many of the body's systems, especially the nerves and blood vessels.

Between 2000 and 2016, there was a 5% increase in premature mortality rates (i.e. before the age of 70) from diabetes. In high-income countries the premature mortality rate due to diabetes decreased from 2000 to 2010 but then increased in 2010-2016. In lower-middle-income countries, the premature mortality rate due to diabetes increased across both periods.

In this notebook, i will do some feature analysis and try to find out the rootcauses

4 Objectives

1. To experiment with different classification methods to see which yields the highest accuracy
2. Classify whether someone has diabetes or not from given features
3. To determine which features are the most indicative of diabetes

5 Dataset

I have used Pima Indians Diabetes Database Kaggle Dataset

The dataset contains below features and labels: 1. Pregnancies 2. Glucose 3. BloodPressure 4. SkinThickness 5. Insulin 6. BMI 7. DiabetesPedigreeFunction 8. Age 9. Outcome

6 Installing Libraries

```
[ ]: import numpy as np # linear algebra
     import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
     import seaborn as sns # for data visualization
     import matplotlib.pyplot as plt # to plot charts
     from collections import Counter
     import os

     # Modeling
     from sklearn.preprocessing import QuantileTransformer
     from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
     from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
      ↪GradientBoostingClassifier, VotingClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.svm import SVC
     from sklearn.model_selection import GridSearchCV, cross_val_score,␣
      ↪StratifiedKFold, learning_curve, train_test_split


     # Directory Structure
     for dirname, _, filenames in os.walk('/kaggle/input'):
         for filename in filenames:
             print(os.path.join(dirname, filename))
```

I have imported most common libraries used in python for machine learning such as Pandas, Seaborn, Matplitlib etc

7 Importing Data

```
[ ]: df = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
```

```
[ ]: # Get familier with dataset structure

     df.info()
```

Excepting BMI and DiabetesPedigreeFunction all the columns are integer. Outcome is the label containing 1 and 0 values. 1 means person has diabetes and 0 mean person is not diabetic

[ ]:
```
df.describe()
```

There are 768 records in the dataset, in which mean age of people is 33

8 Missing Value Analysis

Next, i will cleanup the dataset which is the important part of data science. Missing data can lead to wrong statistics during modeling and predictions.

[ ]:
```
df.describe()
```

[ ]:
```
# Explore missing values

df.isnull().sum()
```

I observed that there is no missing values in dataset however the features like Glucose, BloodPressure, Insulin, SkinThickness has 0 values which is not possible. We have to replace 0 values with either mean or median values of specific column

[ ]:
```
df['Glucose'] = df['Glucose'].replace(0,df['Glucose'].mean())
df.Glucose.value_counts()
```

[ ]:
```
# Correcting missing values in blood pressure

df[df['BloodPressure'] == 0]['BloodPressure'].value_counts()
df['BloodPressure'] = df['BloodPressure'].replace(0,df['BloodPressure'].mean())
```

There are 35 records with 0 BloodPressure in dataset

[ ]:
```
# Correcting missing values in BMI

df[df['BMI'] == 0]['BMI'].value_counts()
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())
```

[ ]:
```
# Correct missing values in Insulin and SkinThickness

df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].
 ↪median())
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())
```

[ ]:
```
# Review dataset statistics

df.describe()
```

Now i have dataset without missing values in features which is good

9 Exploratory Data Analysis

[ ]:
```
# Show top 5 rows
df.head()
```

Correlation

[ ]:
```
plt.figure(figsize=(13,10))
sns.heatmap(df.corr(),annot=True, fmt = ".2f", cmap = "coolwarm")
```

According to observation, features like Pregnancies, Gluecose, BMI, and Age is more correlated with Outcome

Pregnancies

[ ]:
```
# Explore Pregnancies vs Outcome
plt.figure(figsize=(13,6))
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1], color="Red", shade =␣
  ↪True)
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0], ax =g, color="Green",␣
  ↪shade= True)
g.set_xlabel("Pregnancies")
g.set_ylabel("Frequency")
g.legend(["Positive","Negative"])
```

Outcome

[ ]:
```
sns.countplot('Outcome',data=df)
```

There are more people who do not have diabetes in dataset which is around 65% and 35% people has diabetes

[ ]:
```
df
```

Glucose

[ ]:
```
plt.figure(figsize=(10,6))
sns.violinplot(data=df, x="Outcome", y="Glucose",
               split=True, inner="quart", linewidth=1)
```

The chances of diabetes is gradually increasing with level of Glucose

[ ]:
```
# Explore Glucose vs Outcome

plt.figure(figsize=(13,6))
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade = True)
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade=␣
  ↪True)
g.set_xlabel("Glucose")
g.set_ylabel("Frequency")
```

```
g.legend(["Positive","Negative"])
```

Explore Glucose vs BMI vs Age

[ ]:
```
# Glucose vs BMI vs Age

plt.figure(figsize=(20,10))
sns.scatterplot(data=df, x="Glucose", y="BMI", hue="Age", size="Age")
```

As per observation there are some outliers in features. We need to remove outliers in feature engineering

BloodPressure

[ ]:
```
# Explore Age vs Sex, Parch , Pclass and SibSP
g = sns.catplot(y="BloodPressure",x="Outcome",data=df,kind="box")
g.set_ylabels("Blood Pressure")
g.set_xlabels("Outcome")
```

Age vs Outcome

[ ]:
```
# Explore Age

g = sns.catplot(y="Age",x="Outcome",data=df,kind="box")
g.set_ylabels("Age")
g.set_xlabels("Outcome")
```

DiabetesPedigreeFunction

[ ]:
```
sns.set_theme(style="whitegrid")
plt.figure(figsize=(7,5))

sns.boxenplot(x="Outcome", y="DiabetesPedigreeFunction",
              color="b",
              scale="linear", data=df)
g.set_ylabels("Diabetes Pedigree Function")
g.set_xlabels("Outcome")
```

10 Feature Enginnering

Till now, i explored the dataset, did missing value corrections and data visualization. Next, i have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning. Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible.

11 Outlier Detection

```
[ ]: def detect_outliers(df,n,features):
         outlier_indices = []
         """
         Detect outliers from given list of features. It returns a list of the␣
     ↪indices
         according to the observations containing more than n outliers according
         to the Tukey method
         """
         # iterate over features(columns)
         for col in features:
             Q1 = np.percentile(df[col], 25)
             Q3 = np.percentile(df[col],75)
             IQR = Q3 - Q1

             # outlier step
             outlier_step = 1.5 * IQR

             # Determine a list of indices of outliers for feature col
             outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 +␣
     ↪outlier_step )].index

             # append the found outlier indices for col to the list of outlier␣
     ↪indices
             outlier_indices.extend(outlier_list_col)

         # select observations containing more than 2 outliers
         outlier_indices = Counter(outlier_indices)
         multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

         return multiple_outliers

     # detect outliers from numeric features
     outliers_to_drop = detect_outliers(df, 2 ,["Pregnancies", 'Glucose',␣
      ↪'BloodPressure', 'BMI', 'DiabetesPedigreeFunction', 'SkinThickness',␣
      ↪'Insulin', 'Age'])
```

```
[ ]: df.loc[outliers_to_drop] # Show the outliers rows
```

```
[ ]: df.drop(df.loc[outliers_to_drop].index, inplace=True)
```

I have successfully removed all outliers from dataset now. The next step is to split the dataset in train and test and procceed the modeling

12 Modeling

13 Transforming Data

Before i split the dataset i need to transform the data into quantile using sklearn.preprocessing

```
[ ]: q   = QuantileTransformer()
     X = q.fit_transform(df)
     transformedDF = q.transform(X)
     transformedDF = pd.DataFrame(X)
     transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure',␣
      ↪'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age',␣
      ↪'Outcome']
```

```
[ ]: transformedDF.head()
```

14 Data Splitting

```
[ ]: ## Separate train dataset and test dataset
     features = df.drop(["Outcome"], axis=1)
     labels = df["Outcome"]
     x_train, x_test, y_train, y_test = train_test_split(features, labels,␣
      ↪test_size=0.30, random_state=7)
```

15 Cross Validate Models

```
[ ]: def evaluate_model(models):
         """
         Takes a list of models and returns chart of cross validation scores using␣
      ↪mean accuracy
         """

         # Cross validate model with Kfold stratified cross val
         kfold = StratifiedKFold(n_splits = 10)

         result = []
         for model in models :
             result.append(cross_val_score(estimator = model, X = x_train, y =␣
      ↪y_train, scoring = "accuracy", cv = kfold, n_jobs=4))

         cv_means = []
         cv_std = []
         for cv_result in result:
             cv_means.append(cv_result.mean())
             cv_std.append(cv_result.std())
```

```
    result_df = pd.DataFrame({
        "CrossValMeans":cv_means,
        "CrossValerrors": cv_std,
        "Models":[
            "LogisticRegression",
            "DecisionTreeClassifier",
            "AdaBoostClassifier",
            "SVC",
            "RandomForestClassifier",
            "GradientBoostingClassifier",
            "KNeighborsClassifier"
        ]
    })

    # Generate chart
    bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result_df,␣
 ↪orient = "h")
    bar.set_xlabel("Mean Accuracy")
    bar.set_title("Cross validation scores")
    return result_df
```

[ ]:
```
# Modeling step Test differents algorithms
random_state = 30
models = [
    LogisticRegression(random_state = random_state, solver='liblinear'),
    DecisionTreeClassifier(random_state = random_state),
    AdaBoostClassifier(DecisionTreeClassifier(random_state = random_state),␣
 ↪random_state = random_state, learning_rate = 0.2),
    SVC(random_state = random_state),
    RandomForestClassifier(random_state = random_state),
    GradientBoostingClassifier(random_state = random_state),
    KNeighborsClassifier(),
]
evaluate_model(models)
```

As per above observation, i found that SVC, RandomForestClassifier, and LogisticRegression model has more accuracy. Next, i will do hyper parameter tuning on three models

16 Hyperparameter Tuning

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

I have done tuning process for SVC, RandomForestClassifier, and LogisticRegression models one by one

```
[ ]: # Import libraries
     from sklearn.model_selection import GridSearchCV
     from sklearn.metrics import classification_report
```

```
[ ]: def analyze_grid_result(grid_result):
         '''
         Analysis of GridCV result and predicting with test dataset
         Show classification report at last
         '''

         # Best parameters and accuracy
         print("Tuned hyperparameters: (best parameters) ", grid_result.best_params_)
         print("Accuracy :", grid_result.best_score_)

         means = grid_result.cv_results_["mean_test_score"]
         stds = grid_result.cv_results_["std_test_score"]
         for mean, std, params in zip(means, stds, grid_result.
      ↪cv_results_["params"]):
             print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
         print()

         print("Detailed classification report:")
         print()
         y_true, y_pred = y_test, grid_result.predict(x_test)
         print(classification_report(y_true, y_pred))
         print()
```

First of all i have imported GridSearchCV and classification_report from sklearn library. Then, i have defined analyze_grid_result method which will show prediction result. I called this method for each Model used in SearchCV

17 LogisticRegression

```
[ ]: # Define models and parameters for LogisticRegression
     model = LogisticRegression(solver='liblinear')
     solvers = ['newton-cg', 'liblinear']
     penalty = ['l2']
     c_values = [100, 10, 1.0, 0.1, 0.01]

     # Define grid search
     grid = dict(solver = solvers, penalty = penalty, C = c_values)
     cv = StratifiedKFold(n_splits = 50, random_state = 1, shuffle = True)
     grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = cv,␣
      ↪scoring = 'accuracy', error_score = 0)
     logi_result = grid_search.fit(x_train, y_train)
```

```
# Logistic Regression Hyperparameter Result
analyze_grid_result(logi_result)
```

As per my obversation, in LogisticRegression it returned best score 0.78 with {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'} parameters. Next i will perform tuning for other models.

18 SVC

[ ]:
```
# Define models and parameters for LogisticRegression
model = SVC()

# Define grid search
tuned_parameters = [
    {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},
    {"kernel": ["linear"], "C": [1, 10, 100, 1000]},
]
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv␣
↪= cv, scoring = 'accuracy', error_score = 0)
scv_result = grid_search.fit(x_train, y_train)

# SVC Hyperparameter Result
analyze_grid_result(scv_result)
```

SVC Model gave max 0.77 accuracy which is bit less than LogisticRegression. I will not use this model anymore.

19 RandomForestClassifier

[ ]:
```
# Define models and parameters for LogisticRegression
model = RandomForestClassifier(random_state=42)


# Define grid search
tuned_parameters = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv␣
↪= cv, scoring = 'accuracy', error_score = 0)
grid_result = grid_search.fit(x_train, y_train)
```

```
# SVC Hyperparameter  Result
analyze_grid_result(grid_result)
```

Randomforest model gave max 0.76% accuracy which is not best comparing to other model. So i decided to use LogisticRegression Model for prediction

20 Prediction

Till now, i worked on EDA, Feature Engineering, Cross Validation of Models, and Hyperparameter Tuning and find the best working Model for my dataset. Next, I did prediction from my test dataset and storing the result in CSV

[ ]:
```
y_pred = logi_result.predict(x_test)
print(classification_report(y_test, y_pred))
```

[ ]:
```
x_test['pred'] = y_pred
x_test
```