SURYA GROUP OF INSTITUTIONS

VIKRAVANDI-605652

NAAN MUDHALVAN PROJECT

AI-BASED DIABETES PREDICTION SYSTEM

PHASE-4

DEVELOPMENT PART-2

PRESENTED BY

NAME:S.DINESH

REG.NO:422221106006

DEPT:ECE

INTRODUCTION:

In this phase, we use various ML and DL algorithms have been used to predict diabetes in our dataset in this section. we will utilize logistic regression, random forest, decision tree, artificial neural networks and using catboost algorithm etc. algorithms to predict and analyse results and compare these algorithms on the basis of performance.

IMPORT LIBRARIES:

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

Scikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best thing about Keras is that it allows for easy and fast prototyping.

PyTorch is a popular open-source Machine Learning library for Python based on Torch, which is an open-source Machine Learning library that is implemented in C with a wrapper in Lua. It has an extensive choice of tools and libraries that support Computer Vision, Natural Language Processing(NLP), and many more ML programs. It allows developers to perform computations on Tensors with GPU acceleration and also helps in creating computational graphs.

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for grouping, combining and filtering data.

Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc..,

 # Import Dependencies

```
%matplotlib inline

# Start Python Imports
import math, time, datetime

import random as rd

# Data Manipulation
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import missingno as msno
import seaborn as sns
plt.style.use('seaborn-whitegrid')

# Preprocessing
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, label_binarize

# Machine learning
import catboost
from sklearn.model_selection import train_test_split
from sklearn import model_selection, tree, preprocessing, metrics, linear_model
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from catboost import CatBoostClassifier, Pool, cv

# Let's be rebels and ignore warnings for now
import warnings
warnings.filterwarnings('ignore')
```

```
/tmp/ipykernel_20/160898924.py:16: MatplotlibDeprecationWarning: The seaborn s
tyles shipped by Matplotlib are deprecated since 3.6, as they no longer corres
pond to the styles shipped by seaborn. However, they will remain available as
'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
  plt.style.use('seaborn-whitegrid')
```

HELPING FUNCTIONS:

Activation Function: Activation functions introduce non-linearity into the neural networks, allowing them to learn complex patterns. Common activation functions include Sigmoid, Tanh, ReLU (Rectified Linear Unit), and Softmax.

Loss Functions (Cost Functions): Loss functions measure the difference between the predicted values and the actual values (labels) in the training data. They are used to train machine learning models by minimizing the error. Common loss functions include Mean Squared Error (MSE) for regression problems and Cross-Entropy Loss for classification problems.

Optimization Algorithms: Optimization algorithms are used to minimize the loss function during the training of machine learning models. Gradient Descent, Stochastic Gradient Descent (SGD), and variants like Adam and RMSprop are commonly used optimization algorithms.

Regularization Techniques: Regularization methods are used to prevent overfitting, which occurs when a model performs well on the training data but poorly on unseen data. L1 and L2 regularization add penalty terms to the loss function, encouraging the model to use simpler (smoother) solutions. Dropout is another technique where randomly selected neurons are ignored during training to prevent overfitting.

Data Preprocessing: Data preprocessing techniques include data cleaning, feature scaling, and feature engineering. Data is often normalized or standardized to ensure that all features have the same scale. Categorical variables are encoded into numerical values through techniques like one-hot encoding.

Cross-Validation: Cross-validation is a technique used to assess the performance and generalizability of a machine learning model. It involves dividing the dataset into multiple subsets (folds) and training the model on different subsets while evaluating it on the remaining data.

Evaluation Metrics: Evaluation metrics are used to measure the performance of machine learning models. Common evaluation metrics include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC) for classification problems. For regression problems, metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared are used.

Feature Selection: Feature selection techniques help in selecting the most relevant features for the model, improving its performance and reducing overfitting. Common methods include Recursive Feature Elimination (RFE) and feature importance from tree-based models.

Ensemble Methods: Ensemble methods combine predictions from multiple machine learning models to improve overall performance. Bagging, Boosting, and Stacking are popular ensemble techniques.

Dimensionality Reduction: Dimensionality reduction techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) are used to reduce the number of features while preserving the important information, making it easier to visualize and analyze the data.

These helping functions and techniques are fundamental to the practice of machine learning and are applied across various types of machine learning algorithms and models.

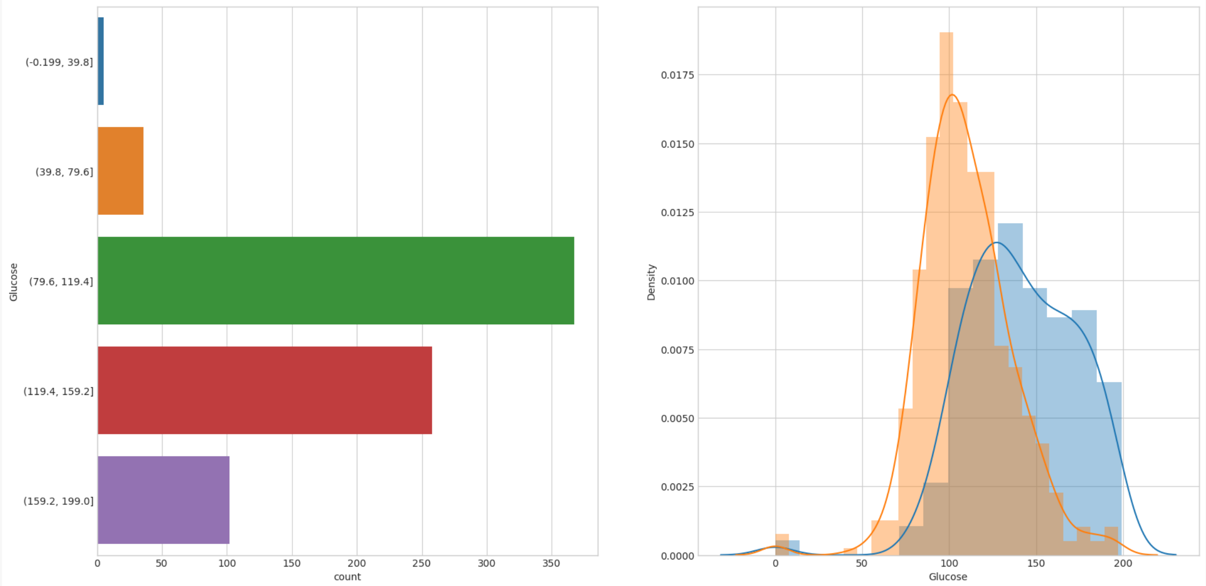
```
def systematic_sample(df, size):
    length = len(df)
    interval = length // size
    rd.seed(None)
    first = rd.randint(0, interval)
    indexes = np.arange(first, length, step = interval)
    return df.iloc[indexes]
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
        return mis_val_table_ren_columns
def fill_na(df):
    for col in df.columns:
        if df[col].isnull().any():
            if df[col].dtypes in ["float", "int"]:
                df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].mode()[0], inplace=True)
def plot_count_dist(data, bin_df, label_column, target_column, figsize=(20, 5)
, use_bin_df=False):
        """
    Function to plot counts and distributions of a label variable and
    target variable side by side.
    ::param_data:: = target dataframe
    ::param_bin_df:: = binned dataframe for countplot
    ::param_label_column:: = binary labelled column
```

```
    ::param_target_column:: = column you want to view counts and distributions
    ::param_figsize:: = size of figure (width, height)
    ::param_use_bin_df:: = whether or not to use the bin_df, default False
    """
    if use_bin_df:
        fig = plt.figure(figsize=figsize)
        plt.subplot(1, 2, 1)
        sns.countplot(y=target_column, data=bin_df);
        plt.subplot(1, 2, 2)
        sns.distplot(data.loc[data[label_column] == 1][target_column],
                     kde_kws={"label": "Survived"});
        sns.distplot(data.loc[data[label_column] == 0][target_column],
                     kde_kws={"label": "Did not survive"});
    else:
        fig = plt.figure(figsize=figsize)
        plt.subplot(1, 2, 1)
        sns.countplot(y=target_column, data=data);
        plt.subplot(1, 2, 2)
        sns.distplot(data.loc[data[label_column] == 1][target_column],
                     kde_kws={"label": "Survived"});
        sns.distplot(data.loc[data[label_column] == 0][target_column],
                     kde_kws={"label": "Did not survive"});
# Function that runs the requested algorithm and returns the accuracy metrics
def fit_ml_algo(algo, X_train, y_train, cv):

    # One Pass
    model = algo.fit(X_train, y_train)
    acc = round(model.score(X_train, y_train) * 100, 2)

    # Cross Validation
    train_pred = model_selection.cross_val_predict(algo,
                                                   X_train,
                                                   y_train,
                                                   cv=cv,
                                                   n_jobs = -1)
    # Cross-validation accuracy metric
    acc_cv = round(metrics.accuracy_score(y_train, train_pred) * 100, 2)

    return train_pred, acc, acc_cv
```

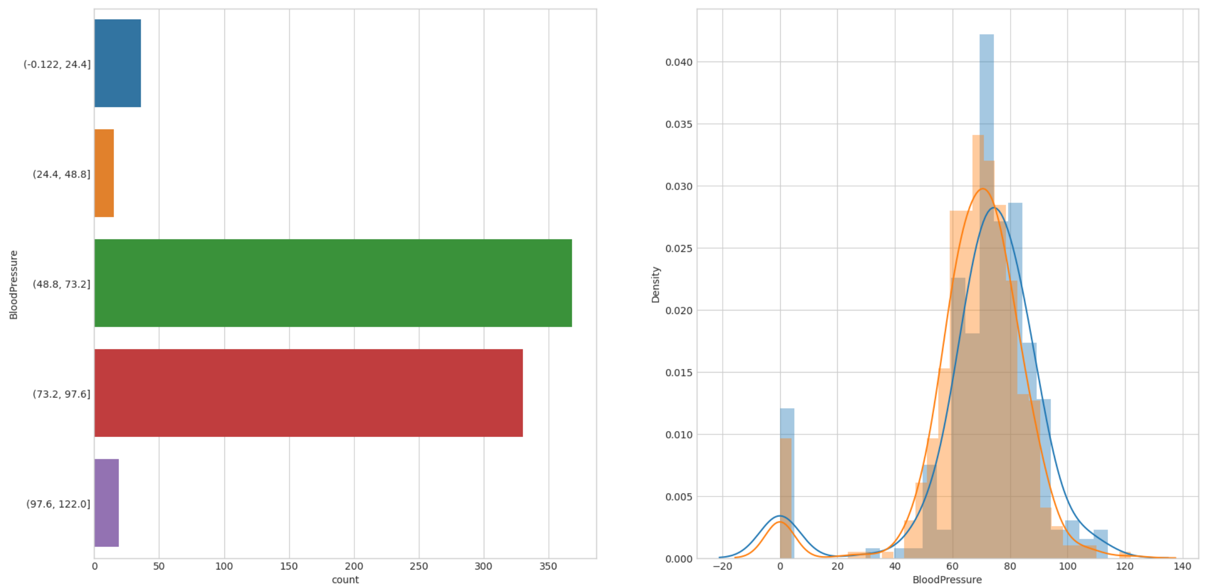
linkcode

```
# Feature Importance
def feature_importance(model, data):
    """
    Function to show which features are most important in the model.
    ::param_model:: Which model to use?
    ::param_data:: What data to use?
    """
    fea_imp = pd.DataFrame({'imp': model.feature_importances_, 'col': data.columns})
    fea_imp = fea_imp.sort_values(['imp', 'col'], ascending=[True, False]).iloc[-30:]
    _ = fea_imp.plot(kind='barh', x='col', y='imp', figsize=(20, 10))
```

```
return fea_imp
#plt.savefig('catboost_feature_importance.png')
```

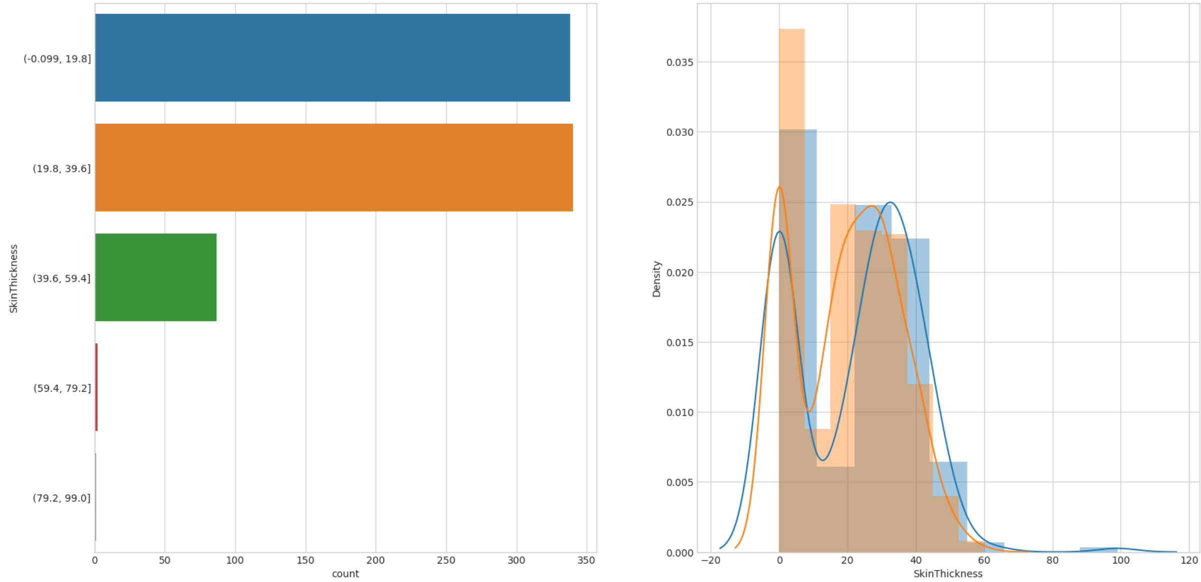



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
                bin_df=df_dis,
                label_column='Outcome',
                target_column='BloodPressure',
                figsize=(20,10),
                use_bin_df=True)
```

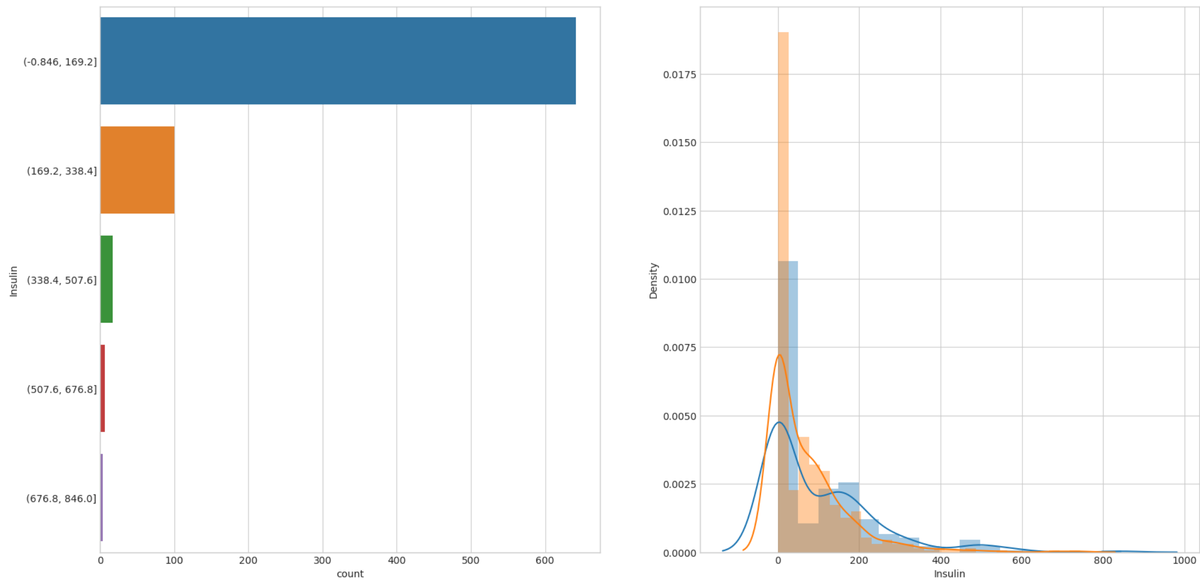



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
```

```
            bin_df=df_dis,
            label_column='Outcome',
            target_column='SkinThickness',

                        figsize=(20,10),
                        use_bin_df=True)
```

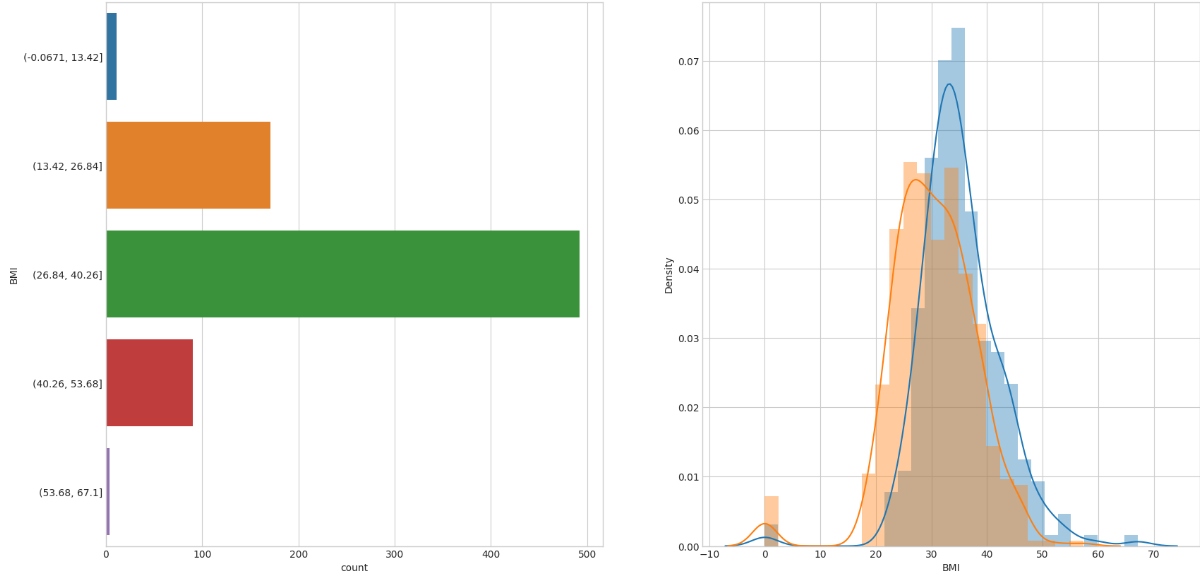



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
            bin_df=df_dis,
            label_column='Outcome',
            target_column='Insulin',
            figsize=(20,10),
            use_bin_df=True)
```

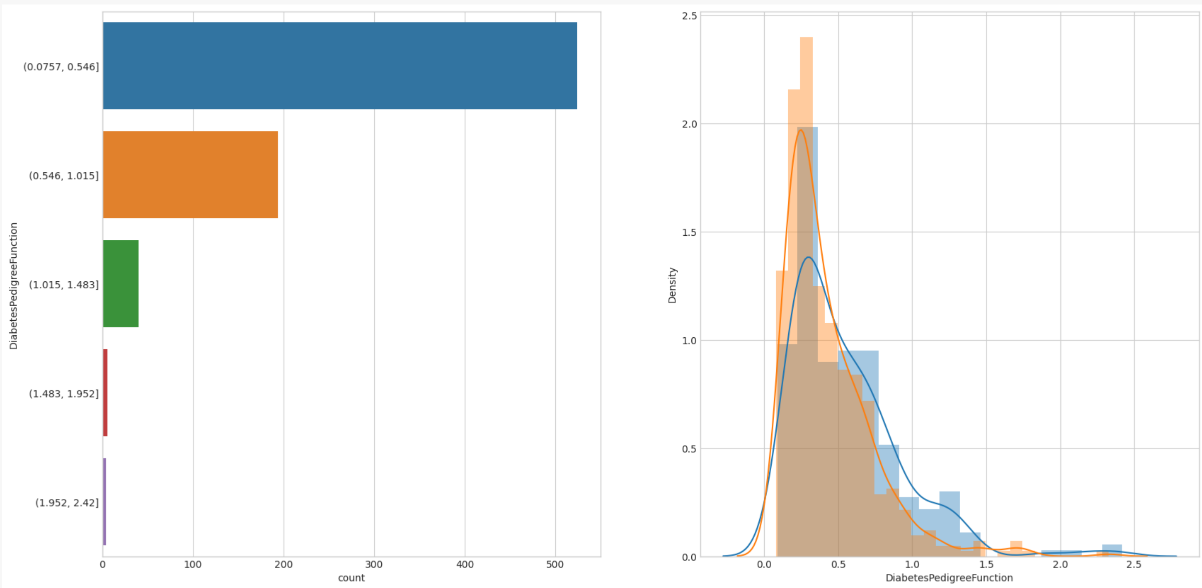



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
            bin_df=df_dis,
            label_column='Outcome',
```

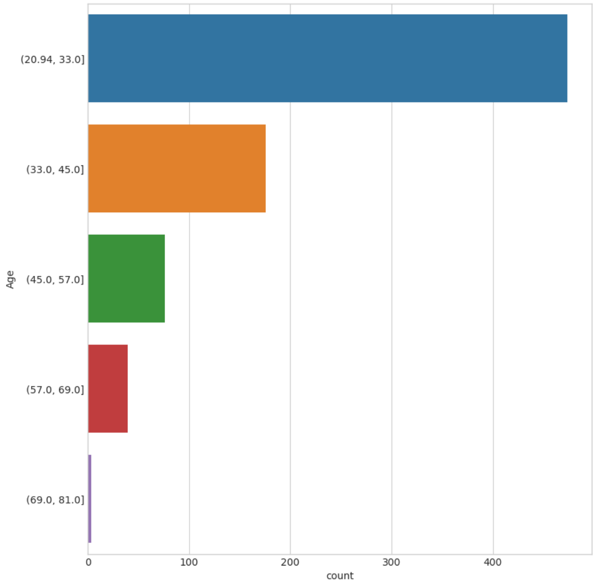
```
                target_column='BMI',
                figsize=(20,10),
                use_bin_df=True)
```

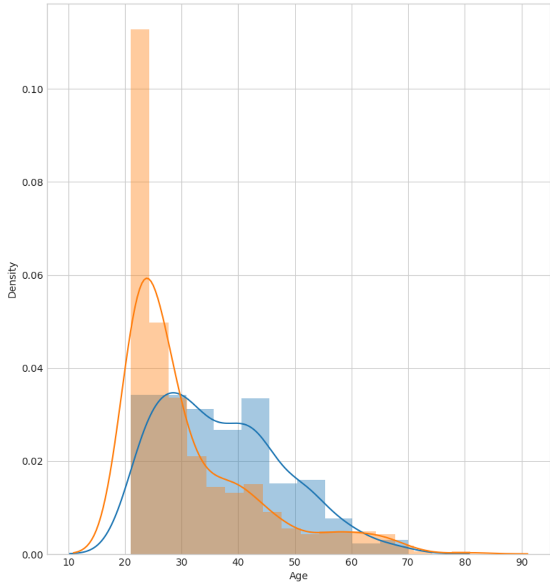



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
                bin_df=df_dis,
                label_column='Outcome',
                target_column='DiabetesPedigreeFunction',
                figsize=(20,10),
                use_bin_df=True)
```




```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
                bin_df=df_dis,
                label_column='Outcome',
                target_column='Age',
                figsize=(20,10),
                use_bin_df=True)
```





431	0	False	False	False	True	False	False	False	False	False	...	False	True	False	False	False	False	True	False	False	False
584	1	False	False	False	False	False	False	False	False	True	...	False	True	False	False	False	False	False	True	False	False
737	0	False	False	False	False	False	False	False	False	True	...	False	True	False	False	False	False	True	False	False	False

```
df_con_enc = pd.get_dummies(df_con, columns=one_hot_cols)
df_con_enc.head()
```

MACHINE LEARNING:

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions, and to uncover key insights in data mining projects.These insights subsequently drive decision making within applications and businesses, ideally impacting key growth metrics. As big data continues to expand and grow, the market demand for data scientists will increase. They will be required to help identify the most relevant business questions and the data to answer them.Machine learning algorithms are typically created using frameworks that accelerate solution development, such as TensorFlow and PyTorch.

1. **A Decision Process**: In general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data.

2. **An Error Function**: An error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.

3. **A Model Optimization Process**: If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this "evaluate and optimize" process, updating weights autonomously until a threshold of accuracy has been met

COMMON MACHINE LEARNING ALGORITHMS:

- **Neural networks:** Neural networks simulate the way the human brain works, with a huge number of linked processing nodes. Neural networks are good at recognizing patterns and play an important role in applications including natural language translation, image recognition, speech recognition, and image creation.

- **Linear regression:** This algorithm is used to predict numerical values, based on a linear relationship between different values. For example, the technique could be used to predict house prices based on historical data for the area.

- **Logistic regression:** This supervised learning algorithm makes predictions for categorical response variables, such as"yes/no" answers to questions. It can be used for applications such as classifying spam and quality control on a production line.

- **Clustering:** Using unsupervised learning, clustering algorithms can identify patterns in data so that it can be grouped. Computers can help data scientists by identifying differences between data items that humans have overlooked.

- **Decision trees:** Decision trees can be used for both predicting numerical values (regression) and classifying data into categories. Decision trees use a branching sequence of linked decisions that can be represented with a tree diagram. One of the advantages of decision trees is that they are easy to validate and audit, unlike the black box of the neural network.

- **Random forests:** In a random forest, the machine learning algorithm predicts a value or category by combining the results from a number of decision trees.

```
# Seclect the dataframe we want to use first for predictions
selected_df = df_con_enc
selected_df.head()
```

```
# Split the dataframe into data and labels
X_train = selected_df.drop('Outcome', axis=1) # data
y_train = selected_df.Outcome # labels
# Shape of the data (without labels)
X_train.shape
```

(768, 1254)
linkcode

```
X_train.head()
# Shape of the labels
y_train.shape
```

(768,)

```
# Logistic Regression
start_time = time.time()
train_pred_log, acc_log, acc_cv_log = fit_ml_algo(LogisticRegression(),
                                                   X_train,
                                                   y_train,
                                                       10)
log_time = (time.time() - start_time)
print("Accuracy: %s" % acc_log)
print("Accuracy CV 10-Fold: %s" % acc_cv_log)
print("Running Time: %s" % datetime.timedelta(seconds=log_time))
```

Accuracy: 96.09
Accuracy CV 10-Fold: 67.45
Running Time: 0:00:02.451250

```
# k-Nearest Neighbours
start_time = time.time()
train_pred_knn, acc_knn, acc_cv_knn = fit_ml_algo(KNeighborsClassifier(),
                                      X_train,
                                      y_train,
                                      10)
knn_time = (time.time() - start_time)
print("Accuracy: %s" % acc_knn)
print("Accuracy CV 10-Fold: %s" % acc_cv_knn)
print("Running Time: %s" % datetime.timedelta(seconds=knn_time))
```

Accuracy: 75.26
Accuracy CV 10-Fold: 66.8
Running Time: 0:00:00.358875

```
# Gaussian Naive Bayes
start_time = time.time()
train_pred_gaussian, acc_gaussian, acc_cv_gaussian = fit_ml_algo(GaussianNB(),
                                                         X_train,
                                                         y_train,
                                                             10)
gaussian_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gaussian)
print("Accuracy CV 10-Fold: %s" % acc_cv_gaussian)
```

```
print("Running Time: %s" % datetime.timedelta(seconds=gaussian_time))
```

```
Accuracy: 94.79
Accuracy CV 10-Fold: 59.9
Running Time: 0:00:00.219412
```

```
# Linear SVC
start_time = time.time()
train_pred_svc, acc_linear_svc, acc_cv_linear_svc = fit_ml_algo(LinearSVC(),
                                                                X_train,
                                                                y_train,
                                                                10)
linear_svc_time = (time.time() - start_time)
print("Accuracy: %s" % acc_linear_svc)
print("Accuracy CV 10-Fold: %s" % acc_cv_linear_svc)
print("Running Time: %s" % datetime.timedelta(seconds=linear_svc_time))
```

```
Accuracy: 100.0
Accuracy CV 10-Fold: 65.23
Running Time: 0:00:00.268296
```

```
# Stochastic Gradient Descent
start_time = time.time()
train_pred_sgd, acc_sgd, acc_cv_sgd = fit_ml_algo(SGDClassifier(),
                                                  X_train,
                                                  y_train,
                                                  10)
sgd_time = (time.time() - start_time)
print("Accuracy: %s" % acc_sgd)
print("Accuracy CV 10-Fold: %s" % acc_cv_sgd)
print("Running Time: %s" % datetime.timedelta(seconds=sgd_time))
```

```
Accuracy: 100.0
Accuracy CV 10-Fold: 63.93
Running Time: 0:00:00.437200
```

```
# Decision Tree Classifier
start_time = time.time()
train_pred_dt, acc_dt, acc_cv_dt = fit_ml_algo(DecisionTreeClassifier(),
                                               X_train,
                                               y_train,
                                              )
dt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_dt)
print("Accuracy CV 10-Fold: %s" % acc_cv_dt)
print("Running Time: %s" % datetime.timedelta(seconds=dt_time))
```

```
Accuracy: 100.0
Accuracy CV 10-Fold: 61.85
Running Time: 0:00:00.596504
```

```
# Gradient Boosting Trees
start_time = time.time()
train_pred_gbt, acc_gbt, acc_cv_gbt = fit_ml_algo(GradientBoostingClassifier(),
                                                  X_train,
                                                  y_train,
                                                  10)
```

gbt_time = (time.time() - start_time)
print("Accuracy: **%s**" % acc_gbt)
print("Accuracy CV 10-Fold: **%s**" % acc_cv_gbt)
print("Running Time: **%s**" % datetime.timedelta(seconds=gbt_time))

Accuracy: 81.9
Accuracy CV 10-Fold: 63.54
Running Time: 0:00:06.448709

CATBOOST ALGORITHM:

CatBoost is a supervised machine learning method that is used by the Train Using AutoML tool and uses decision trees for classification and regression. As its name suggests, CatBoost has two main features, it works with categorical data (the Cat) and it uses gradient boosting (the Boost). Gradient boosting is a process in which many decision trees are constructed iteratively. Each subsequent tree improves the result of the previous tree, leading to better results. CatBoost improves on the original gradient boost method for a faster implementation.

CatBoost overcomes a limitation of other decision tree-based methods in which, typically, the data must be pre-processed to convert categorical string variables to numerical values, one-hot-encodings, and so on. This method can directly consume a combination of categorical and non-categorical explanatory variables without preprocessing. It preprocesses as part of the algorithm. CatBoost uses a method called ordered encoding to encode categorical features. Ordered encoding considers the target statistics from all the rows prior to a data point to calculate a value to replace the categorical feature.Another unique characteristic of CatBoost is that it uses symmetric trees. This means that at every depth level, all the decision nodes use the same split condition.

CatBoost can also be faster than other methods such as XGBoost. It retains certain features—such as cross-validation, regularization, and missing value support—from the prior algorithms. This method performs well with both small data and large data.

```
systematic_sample(X_train, 5)
```

Pregnancies_0	Pregnancies_1	Pregnancies_2	Pregnancies_3	Pregnancies_4	Pregnancies_5	Pregnancies_6	Pregnancies_7	Pregnancies_8	Pregnancies_9	...	Age_63	Age_64	Age_65	Age_66	Age_67	Age_68	Age_69	Age_70	Age_72	Age_81	
42	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False

	Pregnancies_0	Pregnancies_1	Pregnancies_2	Pregnancies_3	Pregnancies_4	Pregnancies_5	Pregnancies_6	Pregnancies_7	Pregnancies_8	Pregnancies_9	...	Age_63	Age_64	Age_65	Age_66	Age_67	Age_68	Age_69	Age_70	Age_72	Age_81
195	False	False	False	False	False	True	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
348	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
501	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
654	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

```
systematic_sample(y_train, 5)
```

```
149    0
302    0
455    1
608    0
761    1
Name: Outcome, dtype: int64
```

```
# Define the categorical features for the CatBoost model
cat_features = np.where(X_train.dtypes != np.float)[0]
cat_features
```

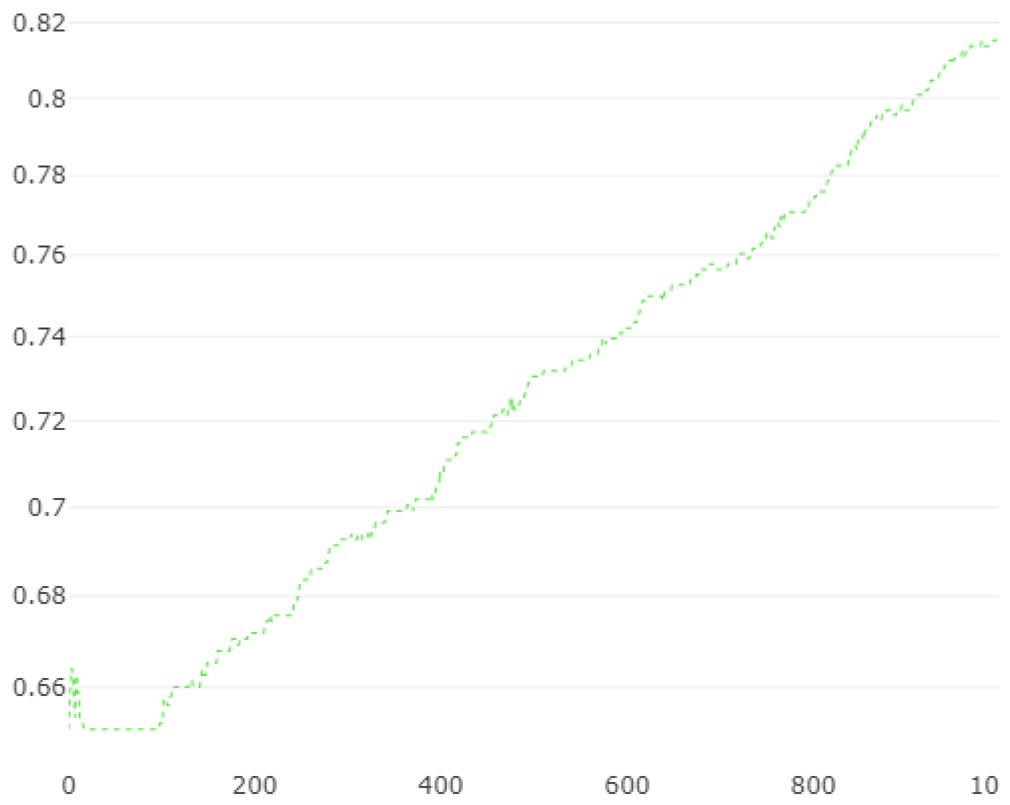
```
array([   0,    1,    2, ..., 1251, 1252, 1253])
```

```
# Use the CatBoost Pool() function to pool together the training data and categori
cal feature labels
train_pool = Pool(X_train,
                  y_train,
                  cat_features)
```

linkcode

```
# CatBoost model definition
```

```
catboost_model = CatBoostClassifier(iterations=1000,
                                    custom_loss=['Accuracy'],
                                    loss_function='Logloss')

# Fit CatBoost model
catboost_model.fit(train_pool,
                   plot=True)

# CatBoost accuracy
acc_catboost = round(catboost_model.score(X_train, y_train) * 100, 2)
```




```
469:	learn: 0.5395230	total: 2.34s	remaining: 2.64s
470:	learn: 0.5394029	total: 2.34s	remaining: 2.63s
471:	learn: 0.5392298	total: 2.35s	remaining: 2.63s
472:	learn: 0.5390315	total: 2.36s	remaining: 2.63s
473:	learn: 0.5388890	total: 2.37s	remaining: 2.63s
474:	learn: 0.5387269	total: 2.37s	remaining: 2.62s
475:	learn: 0.5384723	total: 2.38s	remaining: 2.62s
476:	learn: 0.5383353	total: 2.38s	remaining: 2.62s
477:	learn: 0.5382355	total: 2.39s	remaining: 2.61s
478:	learn: 0.5380677	total: 2.4s	remaining: 2.61s
479:	learn: 0.5378452	total: 2.4s	remaining: 2.61s
480:	learn: 0.5376842	total: 2.41s	remaining: 2.6s
481:	learn: 0.5375600	total: 2.42s	remaining: 2.6s
482:	learn: 0.5374369	total: 2.42s	remaining: 2.6s
483:	learn: 0.5372150	total: 2.43s	remaining: 2.59s
484:	learn: 0.5371141	total: 2.44s	remaining: 2.59s
485:	learn: 0.5369491	total: 2.45s	remaining: 2.59s
```

```
486:	learn: 0.5368111	total: 2.46s	remaining: 2.59s
487:	learn: 0.5367199	total: 2.46s	remaining: 2.59s
488:	learn: 0.5365787	total: 2.47s	remaining: 2.58s
489:	learn: 0.5364203	total: 2.48s	remaining: 2.58s
490:	learn: 0.5362019	total: 2.48s	remaining: 2.57s
491:	learn: 0.5361098	total: 2.49s	remaining: 2.57s
492:	learn: 0.5358471	total: 2.49s	remaining: 2.56s
493:	learn: 0.5357295	total: 2.5s	remaining: 2.56s
494:	learn: 0.5355541	total: 2.5s	remaining: 2.55s
495:	learn: 0.5353691	total: 2.51s	remaining: 2.55s
496:	learn: 0.5352261	total: 2.51s	remaining: 2.54s
497:	learn: 0.5351270	total: 2.52s	remaining: 2.54s
498:	learn: 0.5350399	total: 2.52s	remaining: 2.53s
499:	learn: 0.5349238	total: 2.53s	remaining: 2.53s
500:	learn: 0.5348211	total: 2.53s	remaining: 2.52s
501:	learn: 0.5347660	total: 2.54s	remaining: 2.52s
502:	learn: 0.5346753	total: 2.54s	remaining: 2.51s
503:	learn: 0.5344722	total: 2.55s	remaining: 2.51s
504:	learn: 0.5341997	total: 2.56s	remaining: 2.5s
505:	learn: 0.5340434	total: 2.56s	remaining: 2.5s
```

```
print("---CatBoost Metrics---")
print("Accuracy: {}".format(acc_catboost))
print("Accuracy cross-validation 10-Fold: {}".format(acc_cv_catboost))
print("Running Time: {}".format(datetime.timedelta(seconds=catboost_time)))
```

---CatBoost Metrics---
Accuracy: 81.64
Accuracy cross-validation 10-Fold: 66.54
Running Time: 0:00:59.138368
linkcode

```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression', 'Naive Bayes',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree', 'Gradient Boosting Trees',
              'CatBoost'],
    'Score': [
        acc_knn,
        acc_log,
        acc_gaussian,
        acc_sgd,
        acc_linear_svc,
        acc_dt,
        acc_gbt,

        acc_catboost
    ]})
print("---Reuglar Accuracy Scores---")
models.sort_values(by='Score', ascending=False)
```

---Reuglar Accuracy Scores---

Model	Score	
3	Stochastic Gradient Decent	100.00
4	Linear SVC	100.00
5	Decision Tree	100.00
1	Logistic Regression	96.09
2	Naive Bayes	94.79
6	Gradient Boosting Trees	81.90
7	CatBoost	81.64
0	KNN	75.26

```
cv_models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression', 'Naive Bayes',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree', 'Gradient Boosting Trees',
              'CatBoost'],
    'Score': [
        acc_cv_knn,
        acc_cv_log,
        acc_cv_gaussian,
        acc_cv_sgd,
        acc_cv_linear_svc,
        acc_cv_dt,
        acc_cv_gbt,
        acc_cv_catboost
    ]})
print('---Cross-validation Accuracy Scores---')
```

```
cv_models.sort_values(by='Score', ascending=False)
```

---Cross-validation Accuracy Scores---

Model	Score	
1	Logistic Regression	67.45
0	KNN	66.80
7	CatBoost	66.54
4	Linear SVC	65.23
3	Stochastic Gradient Decent	63.93
6	Gradient Boosting Trees	63.54
5	Decision Tree	61.85
2	Naive Bayes	59.90

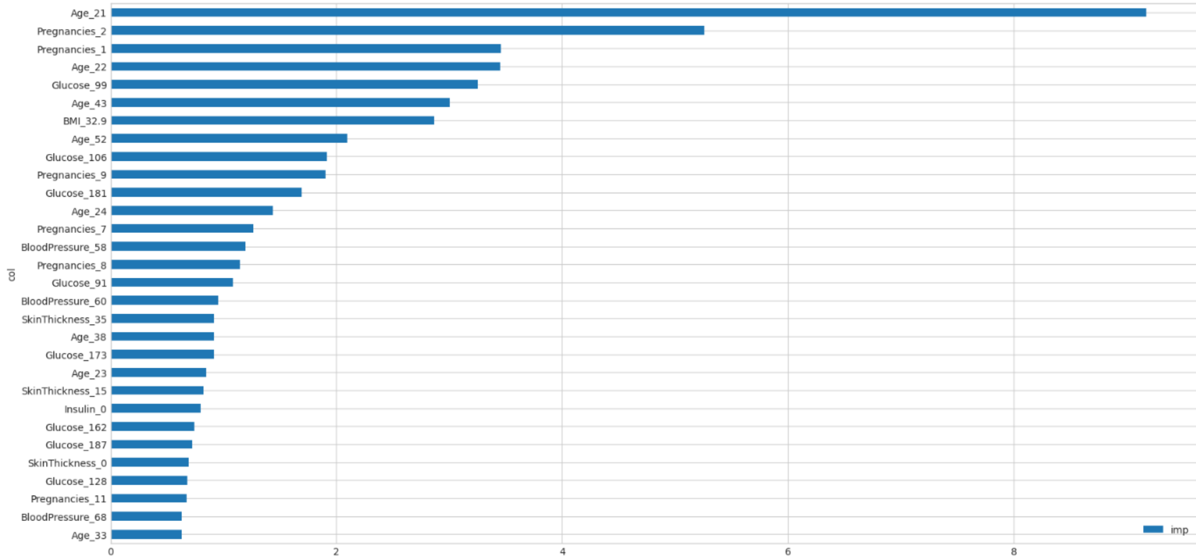
```
# Plot the feature importance scores
feature_importance(catboost_model, X_train)
```

imp	col	
1214	0.628039	Age_33
173	0.628329	BloodPressure_68

imp	col	
11	0.674076	Pregnancies_11
83	0.677186	Glucose_128
200	0.690573	SkinThickness_0
141	0.722829	Glucose_187
117	0.741188	Glucose_162
251	0.797263	Insulin_0
208	0.821834	SkinThickness_15
1204	0.847008	Age_23
128	0.914077	Glucose_173
1219	0.917231	Age_38
228	0.917400	SkinThickness_35
167	0.954864	BloodPressure_60
46	1.084594	Glucose_91

imp	col	
8	1.143536	Pregnancies_8
166	1.197212	BloodPressure_58
7	1.261164	Pregnancies_7
1205	1.437076	Age_24
136	1.694708	Glucose_181
9	1.900946	Pregnancies_9
61	1.916399	Glucose_106
1233	2.096556	Age_52
555	2.866333	BMI_32.9
1224	3.000875	Age_43
54	3.251553	Glucose_99
1203	3.452897	Age_22
1	3.460366	Pregnancies_1

imp	col	
2	5.262568	Pregnancies_2
1202	9.173822	Age_21



CONCLUSION:

We have gone through many more different kind of model and we analyse that the top most efficient algorithm among them till now is the random forest one, that is producing best output and result from all the tried model.