IBM
MACHINE
LEARNING

DEEP AND REINFORCEMENT LEARNING MODELS FOR DIABETES PREDICTION

Fatima, Sayeda

1/13/2022

Table of Contents

1) Project Overview and Data Description

1a) Problem Overview

The drastic and often fatal impacts of diabetes not only are not only worrying but its steady incline expected to reach at 629 million by 2045 has also turned it into global threat (Naz and Ahuja, 2020). Despite its alarming increase, diabetes is principally a preventable disease which can be prevented by adopting healthier lifestyle changes which may also decrease probability of developing other diseases like cancer or heart problems. Hence, early detection of diabetes through a reliable prognosis tool is crucial to either prevent disease onset or stop its further progression.

1b) About the Dataset

1b-i) Brief Description of Chosen Data Set

This project uses a hypothetical dataset 'UCI Pima Diabetes Dataset' which has been acquired for identifying risk of diabetes and was downloaded from the following link:

https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

1b-ii) Summary of Data Attributes

The PIMA dataset exhibits 768 data points (rows) and 9 features (columns) reflecting on patients' characteristics where, based on various factors, each patient has been assigned a Diabetes Score.

Of these, the main features are:

1. times_pregnant,
2. glucose_tolerance_test,
3. blood_pressure,
4. skin_thickness,
5. insulin,
6. bmi,
7. pedigree_function
8. age

1c) Data Exploration, Data Cleansing and Features Engineering

Since the quality of any machine learning model highly depends on quality of data, hence, this stage is not only most important but is also time consuming. Hence, it was conducted in a step-by-step process.

1c-i) Data Exploration

- Data was first loaded into pandas dataframe

Read Dataset

```
## Load in the data set
columns = ["times_pregnant", "glucose_tolerance_test", "blood_pressure", "skin_thickness", "insulin",
           "bmi", "pedigree_function", "age", "has_diabetes"]
features = ["times_pregnant", "glucose_tolerance_test", "blood_pressure", "skin_thickness", "insulin",
           "bmi", "pedigree_function", "age"]

diabetes_df = pd.read_csv(input_file_path, names=columns, header=0)
```

- Column types were then explored

```
Data Rows & Columns: (768, 9)

Data Types:
                           Type
times_pregnant            int64
glucose_tolerance_test    int64
blood_pressure            int64
skin_thickness            int64
insulin                   int64
bmi                     float64
pedigree_function       float64
age                       int64
has_diabetes              int64
```

Data Display:

	times_pregnant	glucose_tolerance_test	blood_pressure	skin_thickness	insulin	bmi	pedigree_function	age	has_diabetes
705	6	80	80	36	0	39.8	0.177	28	0
457	5	86	68	28	71	30.2	0.364	24	0
449	0	120	74	18	63	30.5	0.285	26	0

- Automated Exploratory Data Analysis was performed using Sweetviz

```
display_alert("Initial Exploratory Data Analysis", "success")
display_alert_color("Perform Quick EDA: To see dataset's distribution and its dispersion. ","teal", "warning")
df_eda = sv.analyze(customer_data) # Use Sweetviz for Automated EDA
#df_eda.show_html() # Uncomment to Generate Online Report
display_alert("Method to Generate Preliminary Exploratory Data Analysis ", "warning")
```

Initial Exploratory Data Analysis

Perform Quick EDA: To see dataset's distribution and its dispersion.

Done! Use 'show' commands to display/save. [100%] 00:00 -> (00:00 left)

Report SWEETVIZ_REPORT.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, the report IS saved in your notebook/colab files.

Sweetviz 2.1.4

Get updates, docs & report issues here

Created & maintained by Francois Bertrand

Graphic design by Jean-Francois Hains

DataFrame

NO COMPARISON TARGET

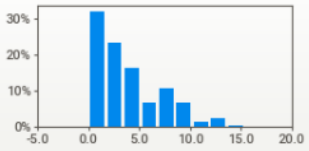
768	ROWS
0	DUPLICATES
55.4 kb	RAM
9	FEATURES
1	CATEGORICAL
8	NUMERICAL
0	TEXT

ASSOCIATIONS

DataFrame

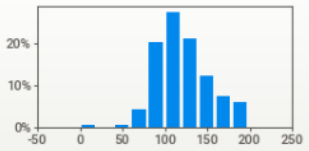
1 times_pregnant

VALUES:	768 (100%)	MAX	17.0	RANGE	17.0		
MISSING:	---	95%	10.0	IQR	5.00		
		Q3	6.0	STD	3.37		
DISTINCT:	17 (2%)	AVG	3.8	VAR	11.4		
		MEDIAN	3.0				
ZEROES:	111 (14%)	Q1	1.0	KURT.	0.159		
		5%	0.0	SKEW	0.902		
		MIN	0.0	SUM	2,953		



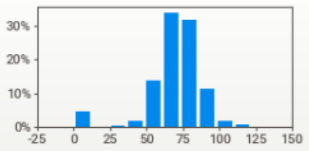
2 glucose_tolerance_test

VALUES:	768 (100%)	MAX	199	RANGE	199		
MISSING:	---	95%	181	IQR	41.2		
		Q3	140	STD	32.0		
DISTINCT:	136 (18%)	AVG	121	VAR	1,022		
		MEDIAN	117				
ZEROES:	5 (<1%)	Q1	99	KURT.	0.641		
		5%	79	SKEW	0.174		
		MIN	0	SUM	92,847		



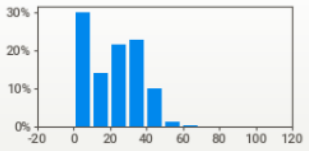
3 blood_pressure

VALUES:	768 (100%)	MAX	122	RANGE	122		
MISSING:	---	95%	90	IQR	18.0		
		Q3	80	STD	19.4		
DISTINCT:	47 (6%)	MEDIAN	72	VAR	375		
		AVG	69				
ZEROES:	35 (5%)	Q1	62	KURT.	5.18		
		5%	39	SKEW	-1.84		
		MIN	0	SUM	53,073		



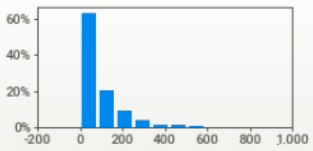
4 skin_thickness

VALUES:	768 (100%)	MAX	99.0	RANGE	99.0		
MISSING:	---	95%	44.0	IQR	32.0		
		Q3	32.0	STD	16.0		
DISTINCT:	51 (7%)	MEDIAN	23.0	VAR	254		
		AVG	20.5				
ZEROES:	227 (30%)	Q1	0.0	KURT.	-0.520		
		5%	0.0	SKEW	0.109		
		MIN	0.0	SUM	15,772		



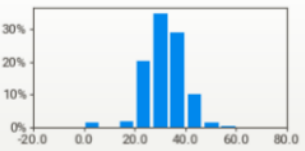
5 insulin

VALUES:	768 (100%)	MAX	846	RANGE	846		
MISSING:	---	95%	293	IQR	127		
		Q3	127	STD	115		
DISTINCT:	186 (24%)	AVG	80	VAR	13,281		
		MEDIAN	30				
ZEROES:	374 (49%)	Q1	0	KURT.	7.21		
		5%	0	SKEW	2.27		
		MIN	0	SUM	61,286		



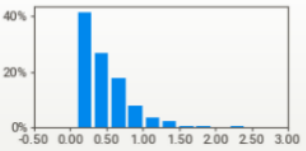
6 bmi

VALUES:	768 (100%)	MAX	67.1	RANGE	67.1		
MISSING:	---	95%	44.4	IQR	9.30		
		Q3	36.6	STD	7.88		
DISTINCT:	248 (32%)	MEDIAN	32.0	VAR	62.2		
		AVG	32.0				
ZEROES:	11 (1%)	Q1	27.3	KURT.	3.29		
		5%	21.8	SKEW	-0.429		
		MIN	0.0	SUM	24,570		



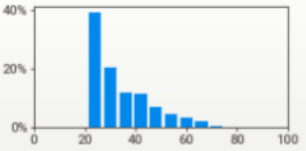
7 pedigree_function

VALUES:	768 (100%)	MAX	2.42	RANGE	2.34		
MISSING:	---	95%	1.13	IQR	0.382		
		Q3	0.63	STD	0.331		
DISTINCT:	517 (67%)	AVG	0.47	VAR	0.110		
		MEDIAN	0.37				
ZEROES:	---	Q1	0.24	KURT.	5.59		
		5%	0.14	SKEW	1.92		
		MIN	0.08	SUM	362		



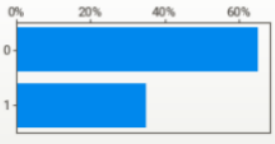
8 age

VALUES:	768 (100%)	MAX	81.0	RANGE	60.0		
MISSING:	---	95%	58.0	IQR	17.0		
		Q3	41.0	STD	11.8		
DISTINCT:	52 (7%)	AVG	33.2	VAR	138		
		MEDIAN	29.0				
ZEROES:	---	Q1	24.0	KURT.	0.643		
		5%	21.0	SKEW	1.13		
		MIN	21.0	SUM	25,529		



9 has_diabetes

VALUES:	768 (100%)	
MISSING:	---	
DISTINCT:	2 (<1%)	



- Additional descriptive statistics were computed to summarize shape of a dataset's distribution, its dispersion and central tendency

Summary Statistics

	count	mean	std	min	25%	50%	75%	max
times_pregnant	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
glucose_tolerance_test	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
blood_pressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
skin_thickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
bmi	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
pedigree_function	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
has_diabetes	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

1c-ii) Data Cleansing Actions & Features Engineering

In machine learning, feature selection is the method to reduce the number of input variables during developing predictive modelling. This reduction in input variables is necessary not only to minimize computational cost of modeling but also to achieve performance improvement of the model.

Among widely practices feature selection approaches include statistical-based feature selection methods which use statistical measures to evaluate relationship between each input variable and the target variable and then select those exhibiting strongest relationship with the latter. While these methods can be both speedy and effective, however, the ultimate choice of statistical measure is largely dependent on data types of both of these variables.

Irrespective of the statistical measure being employed, two dominant feature selection techniques, that is supervised and unsupervised, exist where the former can be further categorized into wrapper, filter and intrinsic techniques. Filter-based feature selection methods employs statistical measures to evaluate correlation between input and output variables so that those exhibiting highest correlations are selected. Statistical measures employed in filter-based feature selection are normally univariate in nature since they evaluate relationship of single input variables one by one with target variable, disregarding their interaction with each other.

Consequently, adopting filter-based feature selection methods, the project approached filter engineering in the following steps.

An automated data cleansing method was created to do the following:

- Drop Columns with Unique Values Less than threshold of 2
- Drop Highly Skewed & Low Correlation Columns with target
- Drop Columns with High Nan Values

Method to Drop Columns

1) With Distinct < 2
2) With High Skewness and Low Correlation to Target
3) Drop Columns With High Nan Values

```
def drop_cols(*name):
    n = name # Extract Dataframe by Name...this will create a 3d tuple
    n = (n[0]) # Convert Tuple to To Dataframe
    df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe

    n = n.fillna(0) #Fill Remaining Missing Values with Zero
    # Find Mean of Null, Nan and Zero Values Before Any Drops
    m0 = n.isin([' ','NULL','NaN', 0]).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_posit

    #  1) Drop Columns with Unique Values Less than threshold
    unique_counts = pd.DataFrame.from_records([(col, n[col].nunique()) for col in n.columns], # get unique counts
                            columns=['Column_Name', 'Unique']).sort_values(by=['Unique'])
    unique = unique_counts[(unique_counts['Unique'] < 2)] #If threshold is lesss than 2 then
    drop1 = (unique['Column_Name'].tolist()) # First List of columns to drop

    print(colored("\nDrop 1: ", 'blue', attrs=['bold'])
          +colored(drop1, 'magenta', attrs=['bold']))

    # 2) Drop Highly Skewed & Low Correlation Columns with target
    drop2 = analyze[(analyze[column] != 1.000000) & ((analyze['skewness'] > 1) & (analyze[column] < 0))]
    print(colored("\nDrop 2: \n ", 'blue', attrs=['bold'])
          +colored(drop2, 'magenta', attrs=['bold']))

    drop2 = drop2.sort_values(by='Columns', ascending=True)
    drop2 = drop2['Columns'].tolist() # Second List of columns to drop

    drop = drop1 + drop2 + delete_features # Final List of columns to drop
    print(colored("\nFinal Column Drop List: ", 'blue', attrs=['bold'])
          +colored(drop, 'magenta', attrs=['bold']))

    if target in drop: # Remove Target from List
        drop = drop.remove(target)
    else:
        print(target, " Not Found.")

    n = n.drop(columns=[col for col in n if col in drop]) # Drop Dataframe Columns if in List

    # Find Mean of Null, Nan and Zero Values Before Dropping
    m1 = n.isin([' ','NULL','NaN', 0]).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_posit

    # 3) Drop Columns With High Nan Values
    #drop_thresh = .90 # Identify Drop Threshold
    #n = n.loc[:, df.isin([' ','NULL', 'NaN',0]).mean() > drop_thresh] # drop columns if Mean is > 0.90

    n = n.fillna(0) #Fill Remaining Missing Values with Zero
    #n = n.replace(["NaN"], 0).sort_values(by=target, ascending=False) # Replace all Nan Values with Zero

    # Find Mean of Null, Nan and Zero Values After Dropping
    m2 = n.isin([' ','NULL','NaN']).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_positior

    #Print Results
    print(colored("\nDataframe Average Null Values Before Any Drops\n ", 'blue', attrs=['bold'])
          +colored(m0, 'magenta', attrs=['bold'])
          +colored("\n\n Low Distict Columns to Drop: ", 'green', attrs=['bold'])
          + colored(drop1, 'red', attrs=['bold'])
          +colored("\n\nDataframe Average Null Values After Dropping Highly Skewed Columns\n ", 'green', attrs=['bold'])
          +colored(m1, 'red', attrs=['bold'])
          +colored("\n\n Drop Columns if Mean is > 0.90 \n", 'green', attrs=['bold'])
          + colored("\nDataframe Average Null Values After Drop and 'Nan' Replacement\n", 'blue', attrs=['bold'])
          +colored(m2, 'magenta', attrs=['bold'])
          +colored(type(m2), 'magenta', attrs=['bold'])
         )
    return n

# Return Function
n = drop_cols(df)
df = n.copy()
#df.info()
```

Null values were summed and were automatically managed by the above function.

```
Dataframe Average Null Values Before Any Drops
 has_diabetes               0.651042
insulin                     0.486979
skin_thickness              0.295573
times_pregnant              0.144531
blood_pressure              0.045573
bmi                         0.014323
glucose_tolerance_test      0.006510
age                         0.000000
pedigree_function           0.000000
dtype: float64

 Low Distict Columns to Drop: []

Dataframe Average Null Values After Dropping Highly Skewed Columns
 has_diabetes               0.651042
insulin                     0.486979
skin_thickness              0.295573
times_pregnant              0.144531
blood_pressure              0.045573
bmi                         0.014323
glucose_tolerance_test      0.006510
age                         0.000000
pedigree_function           0.000000
dtype: float64

 Drop Columns if Mean is > 0.90

Dataframe Average Null Values After Drop and 'Nan' Replacement
has_diabetes                0.0
age                         0.0
pedigree_function           0.0
bmi                         0.0
insulin                     0.0
skin_thickness              0.0
blood_pressure              0.0
glucose_tolerance_test      0.0
times_pregnant              0.0
dtype: float64<class 'pandas.core.series.Series'>
```

Outlier Treatment: Like Supervised learning, deep learning models are also sensitive to outliers. Hence, an automated method was created to replace outliers with "Mode", that is the most common value.

Method to Explore and Adjust Outliers:

Replace Outlier Values with Mode (Most Frequent Value)

```
def outliers(*name):
    n = name # Extract Dataframe by Name...this will create a 3d tuple
    n = (n[0]) # Convert Tuple to To Dataframe
    df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe

    cols = n.columns # All Columns

    # Numeric Columns
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    numeric_cols = n.select_dtypes(include=numerics)
    numeric_cols = numeric_cols.columns.tolist()

    # Object Columns
    categorical_cols = list(set(cols) - set(numeric_cols))

    # Skewed Columns
    skewed_cols = analyze[(analyze['skewness'] > 0) | (analyze['skewness'] < 0)]
    skewed_cols = skewed_cols['Columns'].tolist()
    skewed_cols.remove(target) # Remove target column

    # Replace Outliers
    for col in n.columns:
        if col in skewed_cols:
            print(colored(col, 'magenta', attrs=['bold'])
                  + colored(" is Skewed... ", 'blue', attrs=['bold'])
                 )

            mode = n[col].mode()
            mode = mode[0]

            if col in numeric_cols:
                print(colored(col, 'magenta', attrs=['bold'])
                      + colored(" Column Type is: ", 'blue', attrs=['bold'])
                      + colored(n[col].dtypes, 'red', attrs=['bold'])
                     )
                #Calculate quantiles and IQR
                Q1 = n[col].quantile(0.25) # Same as np.percentile but maps (0,1) and not (0,100)
                Q3 = n[col].quantile(0.75)
                IQR = Q3 - Q1
                # Replace with Mode
                n[col] = np.where((n[col] < (Q1 - 1.5 * IQR)) | (n[col] > (Q3 + 1.5 * IQR)), mode, n[col])

                print(colored("\nReplaced ", 'blue', attrs=['bold'])
                      +colored(col, 'magenta', attrs=['bold'])
                      + colored(" Skewed Values by Mode: ", 'blue', attrs=['bold'])
                      + colored(mode, 'red', attrs=['bold'])
                      + colored('\n', 'magenta', attrs=['bold'])
                      + colored((n[col]), 'green', attrs=['bold'])
                     )

            else:
                print("")
    df_transformed = n.copy()
    return df_transformed
```

Apply 'outliers' Method to New Dataframe

```
times_pregnant is Skewed...
times_pregnant Column Type is: int64

Replaced times_pregnant Skewed Values by Mode: 1
0       6
1       1
2       8
3       1
4       0
       ..
763    10
764     2
765     5
766     1
767     1
```

Data Splitting & Normalization: To speed up algorithms' learning, data normalization was carried out.

Provide Method to Split Training and Testing Dataset

```
def split_data(*name):
    n = name # Extract Dataframe by Name...this will create a 3d tuple
    n = (n[0]) # Convert Tuple to To Dataframe
    df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe to print later
    X = n.loc[ : , n.columns != target] # Remove Target Column
    y = n[target].astype('int')
    return train_test_split(X, y, test_size=0.25, stratify=y, random_state = rs)

display_alert_color_4("Split Data into Test & Train Set: ","darkgreen", "warning")
```

Split Data into Test & Train Set:

```
X_train, X_test, y_train, y_test = split_data(df_transformed) # Call the Method to Split Training and Testing Dataset

# Normalize Data
normalizer = StandardScaler()
X_train_norm = normalizer.fit_transform(X_train)
X_test_norm = normalizer.transform(X_test)

display_alert_color_4("Calculate Mean to Check Proportion of Positive Values: ","darkbrown", "warning")
```

2) Main Objectives of the Analysis

Even though healthcare organizations normally collect big data including electronic health records and images, nevertheless, its robust analysis to acquire meaningful and reliable insights remains a key challenge. Given promising results evidenced by deep and other machine learning methodologies, such medical data can be automatically analysed to discover hidden patterns and factors which may aid in diabetes diagnosis at an early stage.

2a) Primary Objective

Hence, the main objective of this project is to present an automated methodology for employing different variations of Keras deep learning model for diabetes prediction using the PIMA dataset.

2a) Secondary Objectives

Secondary objective are as follows:

1. Build a baseline performance using Random Forest Model (RFM) for comparison
2. Develop automated grid search method for selecting best hyperparameters for Keras model development
3. Evaluate Keras results and compare with those of RFM to validate model robustness

3) Summary of Training Different Deep Learning Models

3a) Keras Hyperparameter Tuning using Grid Search

An automated method was created to find optimal parameters for various Keras Optimizers:

GRID Search for Hyperparameter Tuning

Get Script Start Time

```
Script Start Time: 2023-01-09 18:47:23.204268
```

```python
# Function to create model, required for KerasClassifier
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(6, input_shape=(int(len(X.columns)),), activation='relu'))
    model.add(Dense(6, input_shape=(int(len(X.columns)),), activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# fix random seed for reproducibility
#seed = 7
#tf.random.set_seed(seed)

# create model
model = KerasClassifier(model=create_model, verbose=0)

# define the grid search parameters
batch_size = [10, 20, 40]
epochs = [500, 800, 900]
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']

# Search Best Parameters
param_grid = dict(batch_size=batch_size, epochs=epochs, optimizer=optimizer)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)

# Create Model
model = KerasClassifier(model=create_model, verbose=0)

# Fit Model
grid_result = grid.fit(X, y)
```

3b) Automated Model Building

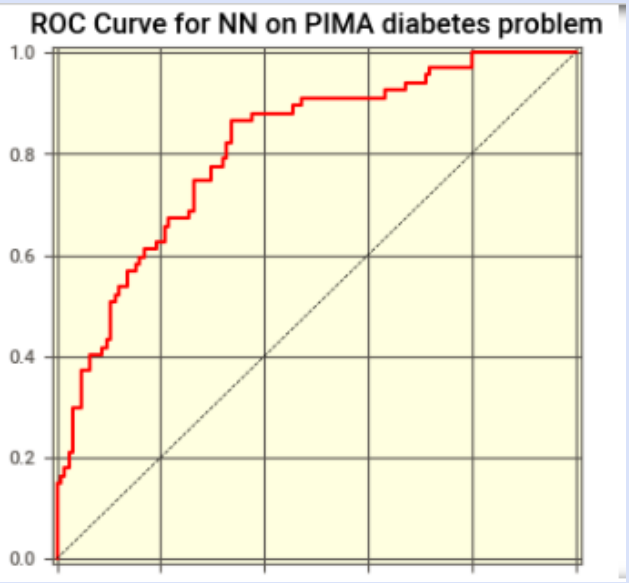
```
optimizer_list = result_df['optimizer'].unique()
#optimizer_list = optimizer_list[:1] # Get the number of models from list...use 1 if only the top performing model is required, however, it may not always give best results
print(colored((" Building Keras for Optimizers: " + optimizer_list), 'blue', attrs=['bold']))

for i in optimizer_list:
    opt = result_df.loc[result_df["optimizer"] == i]
    b_opt = int(opt['batch_size'].iloc[0])
    e_opt = int(opt['epochs'].iloc[0])

    # Create model
    print(colored(("\n" + str(i) + " Model with Batch Size: " + str(b_opt) + " and Epochs: " + str(e_opt)), 'magenta', attrs=['bold'])
          +colored('\nCreating Model', 'green', attrs=['bold']))
    model = Sequential()
    model.add(Dense(6, input_shape=(c,), activation="relu"))
    model.add(Dense(6,  activation="relu"))
    model.add(Dense(1, activation="sigmoid"))

    # Compile model
    model.compile(optimizer=i, loss='binary_crossentropy', metrics=['accuracy'])

    # Set Callbacks
    #   Simple Early Stopping to stop training As soon as the loss of the model begins to increase on the test dataset
    es = EarlyStopping(monitor='val_loss', mode='min', patience=(e_opt/4), verbose=0)# Add patience=(e_opt/4) to set stop
    # Model Checkpoint: The mode parameter controls whether the ModelCheckpoint should be looking for values that minimize our metric or maximize it.
    #   Since we are working with loss, lower is better, so we set mode="min". If we were instead working with val_acc, we would set mode="max" (since higher accuracy is better).
    mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', verbose=0, save_best_only=True)

    print(colored('Fitting Model', 'blue', attrs=['bold']))
    # the fit function returns the run history and is very convenient, as it contains information about the model fit, iterations etc.
    #By setting verbose 0, 1 or 2 you just say how do you want to 'see' the training progress for each epoch.
        #verbose=0 will show you nothing (silent)
        #verbose=1 will show you an animated progress bar =======
        #verbose=2 will just mention the number of epoch [Epoch 1/10]

    run_hist =  model.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test), epochs=e_opt, batch_size=b_opt, verbose=0, callbacks=[es, mc])

    # Let's look at the run_hist_1 object that was created, specifically its history attribute.
    ##run_hist.history.keys()

    print(colored('Making Model Predictions', 'magenta', attrs=['bold']))
    y_pred_prob_nn = model.predict(X_test_norm)#.ravel()
    y_pred_class_nn = np.argmax(y_pred_prob_nn,axis=0)


    print(colored('Calculating AUC Score', 'magenta', attrs=['bold']))
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_nn)
    auc_score = auc(fpr, tpr)
    print(colored((str(i)+' Model auc_score :'+ str(auc_score)), 'magenta', attrs=['bold']))

    # Print model validation loss performance
    print(colored((i+': Computing Model validation loss performance'), 'magenta', attrs=['bold']))
    m = (run_hist.history)
    m = pd.DataFrame.from_dict(m, orient='columns')
    print(colored(m.tail(1), 'magenta', attrs=['bold']))
    # Get val_loss value for head and tail
    vlh = ((m["val_loss"].head(1).iloc[0]))
    vlt = ((m["val_loss"].tail(1).iloc[0]))
    vah = ((m["val_accuracy"].head(1).iloc[0]))
    vat = ((m["val_accuracy"].tail(1).iloc[0]))
    e_act = (len(run_hist.history["loss"]))

    # initialize data of lists.
    data = {'Optimizer': [i],
            'AUC Score': [auc_score],
            'Input Val Loss': [vlh],
            'Output Val Loss': [vlt],
            'Input Val Accuracy': [vah],
            'Output Val Accuracy': [vat],
            'Epochs': [e_opt],
            'Actual Epochs': [e_act]}
    # Create DataFrame
    m = pd.DataFrame(data)
    res = res.append(m, ignore_index=True)
    print(colored(res, 'blue', attrs=['bold']))

    # Get Feature Importance
    e = shap.KernelExplainer(model, X_train_norm)
    shap_values = e.shap_values(X_test_norm)

    # Get Feature Importance in Dataframe
    shap_mean = np.abs(shap_values).mean(axis=0)
    shap_mean = pd.DataFrame(shap_mean, columns = features)
    shap_mean = shap_mean.mean()
    shap_df = shap_mean.to_frame().reset_index()
    shap_df = shap_df.rename({'index':'Features', 0: 'Importance'}, axis=1)#.sort_values(by='Importance', ascending=True)
    shap_df['Importance'] = abs(shap_df['Importance'])
    shap_df['Model'] = (i)
    shap_df = shap_df.sort_values(by='Importance', ascending=False)
    model_features = model_features.append(shap_df, ignore_index=True)

    print(colored((i+': List Feature Importance\n'), 'magenta', attrs=['bold'])
          +colored(shap_df, 'blue', attrs=['bold']))

    # Plot the roc curve
    print(colored(("\n'+i+': ROC Curve:"), 'magenta', attrs=['bold']))
    plot_roc(y_test, y_pred_prob_nn, 'NN')

    # Let's plot the training loss and the validation loss over the different epochs and see how it looks.
    fig, (ax1, ax2) = plt.subplots(2, 1)
    ax1.plot(run_hist.history["loss"],'r', marker='.', label="Train Loss")
    ax1.plot(run_hist.history["val_loss"],'b', marker='.', label="Validation Loss")
    ax1.set(title= (i + ": Displaying Model Validation Loss").format(i))
    ax1.legend()
    ax2.set(title= (i + ": Displaying SHAP Feature Importance").format(i))

    ##We can use the shapely values to interpret our model. 'force_plot' showing how each feature influences the output.
    x1 = shap.summary_plot(shap_values[0], X_test, feature_names=features)
    x4 = shap.summary_plot(shap_values, X_test, feature_names=features, plot_type = 'bar', title=(i + ": Displaying SHAP Feature Importance with Bars"))

    # We can use the shapely values to interpret our model.
    shap.initjs()
    # visualize the first prediction's explanation with a force plot
    #shap.force_plot(e.expected_value[0], shap_values[0][0], features = features)
    shap.plots.force(e.expected_value[0], shap_values[0][0],features = features, matplotlib=True, show=False)
    plt.title(i + " Model Feature Importance", y=1.75)
    plt.show()
```

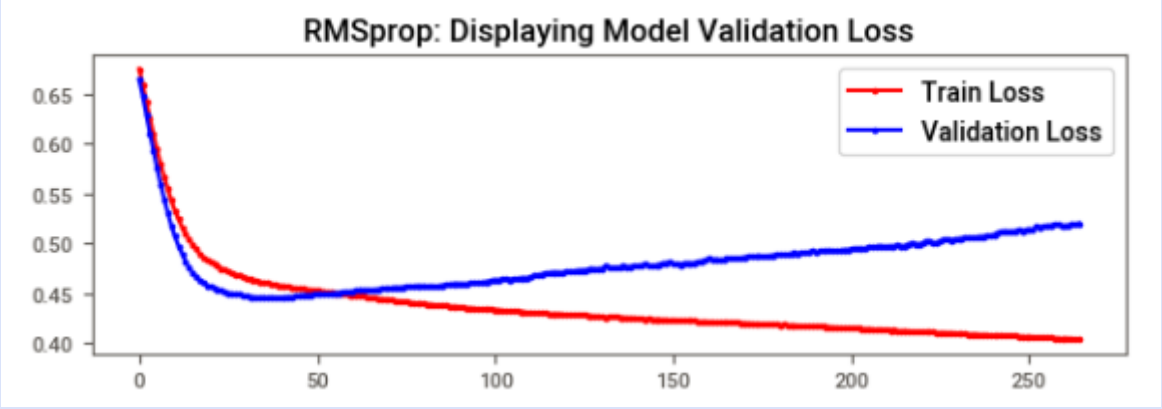
3c) Summarizing Employed Models

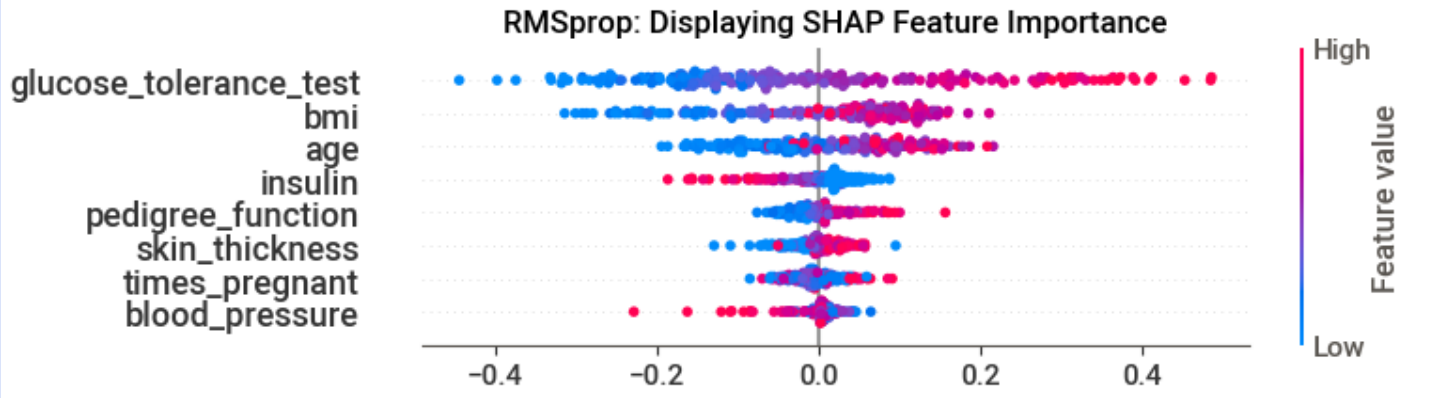
Following is a summary of all Keras Model Variations which have been used to predict onset of diabetes.

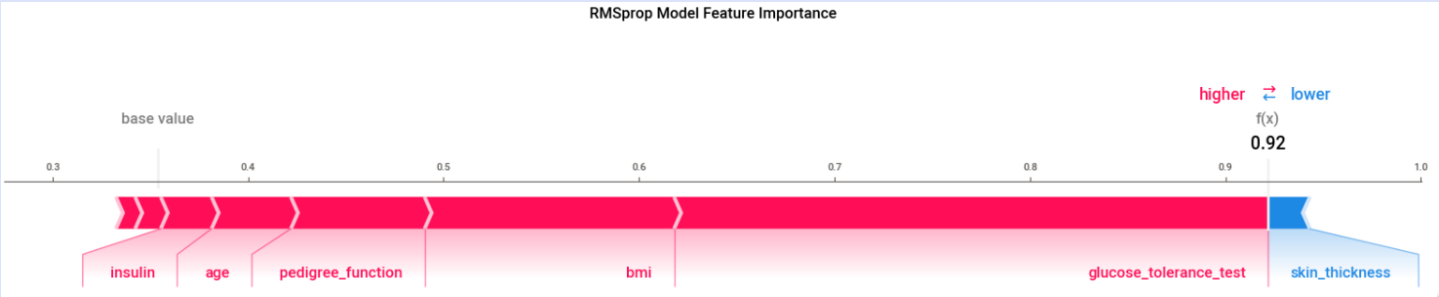
1) Keras Algorithm with Optimizer RMSprop:

```
RMSprop Model with Batch Size: 10 and Epochs: 900
Creating Model
Fitting Model
Using 576 background data samples could cause slower run times. Consider
Making Model Predictions
Calculating AUC Score
RMSprop Model auc_score :0.8108656716417909
RMSprop: Computing Model validation loss performance
         loss  accuracy  val_loss  val_accuracy
264  0.403722  0.814236  0.519842      0.744792
  Optimizer  Input Val Loss  Output Val Loss  Input Val Accuracy  \
0         0        0.000000         0.000000            0.000000
1   RMSprop        0.664082         0.519842            0.697917

   Output Val Accuracy  AUC Score  Epochs  Actual Epochs
0             0.000000        NaN     NaN            NaN
1             0.744792   0.810866   900.0          265.0
```







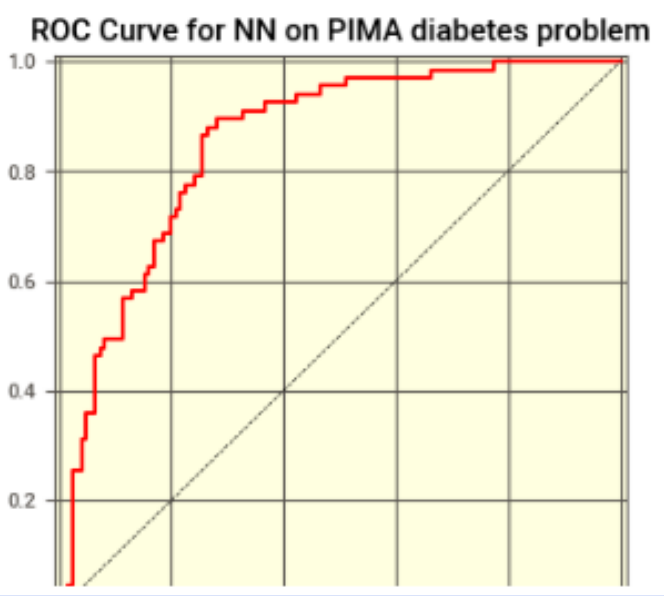


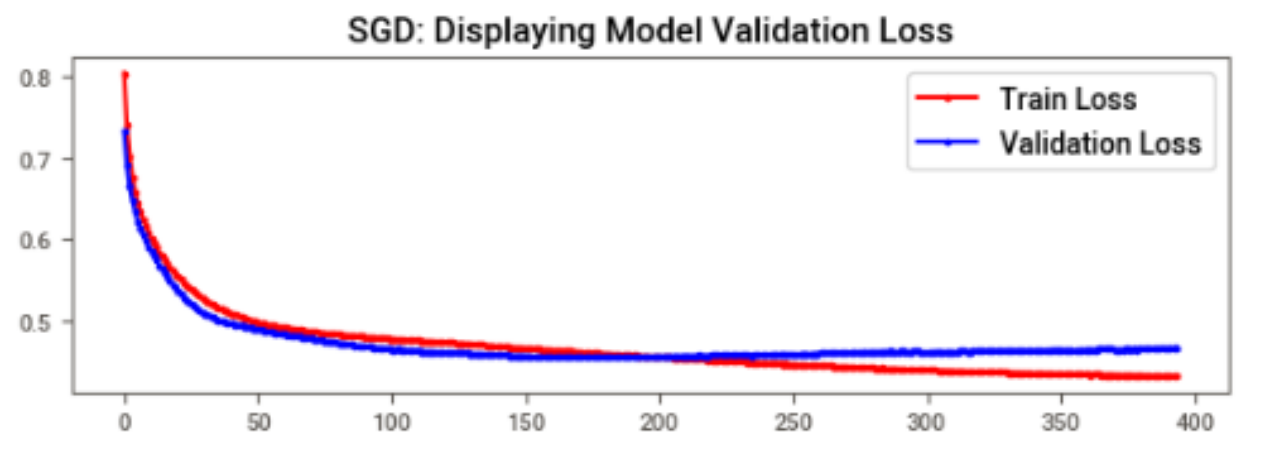
RMSprop: List Feature Importance

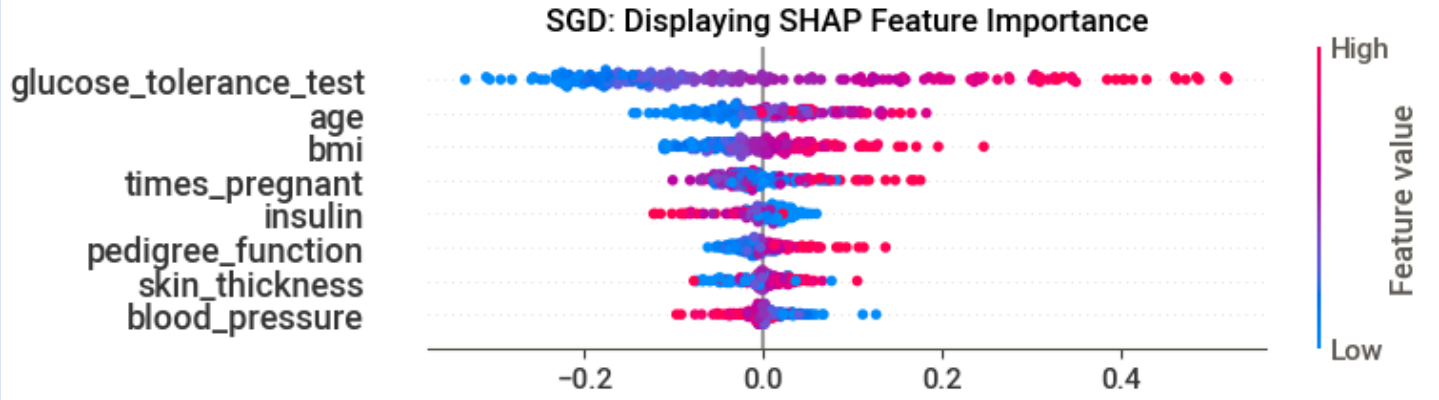
	Features	Importance	Model
1	glucose_tolerance_test	0.168594	RMSprop
5	bmi	0.098901	RMSprop
7	age	0.075343	RMSprop
4	insulin	0.035594	RMSprop
6	pedigree_function	0.024697	RMSprop
3	skin_thickness	0.022470	RMSprop
0	times_pregnant	0.021662	RMSprop
2	blood_pressure	0.017175	RMSprop

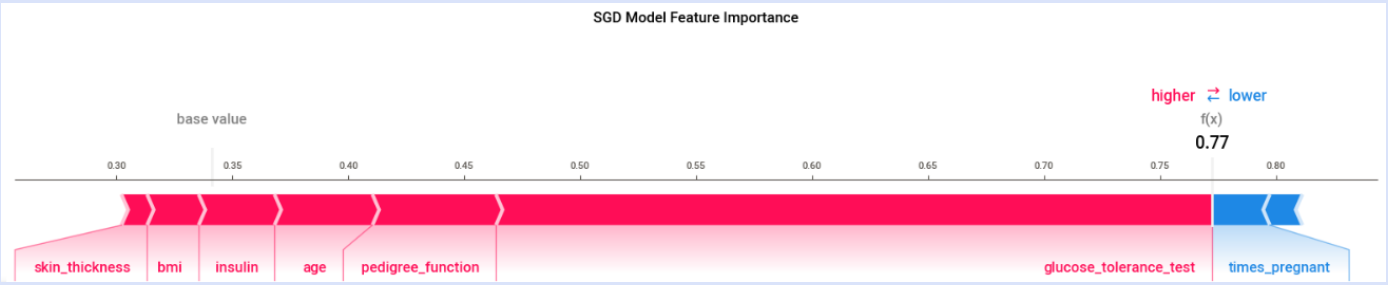
2) Keras Algorithm with Optimizer SGD:

```
SGD Model with Batch Size: 20 and Epochs: 800
Creating Model
Fitting Model

Using 576 background data samples could cause slower run times. Consider

Making Model Predictions
Calculating AUC Score
SGD Model auc_score :0.8511044776119403
SGD: Computing Model validation loss performance
         loss  accuracy  val_loss  val_accuracy
393  0.432101  0.814236   0.46756      0.765625
  Optimizer  Input Val Loss  Output Val Loss  Input Val Accuracy  \
0         0        0.000000         0.000000            0.000000
1   RMSprop        0.664082         0.519842            0.697917
2       SGD        0.732052         0.467560            0.572917

   Output Val Accuracy  AUC Score  Epochs  Actual Epochs
0             0.000000        NaN     NaN            NaN
1             0.744792   0.810866   900.0          265.0
```









SGD: List Feature Importance

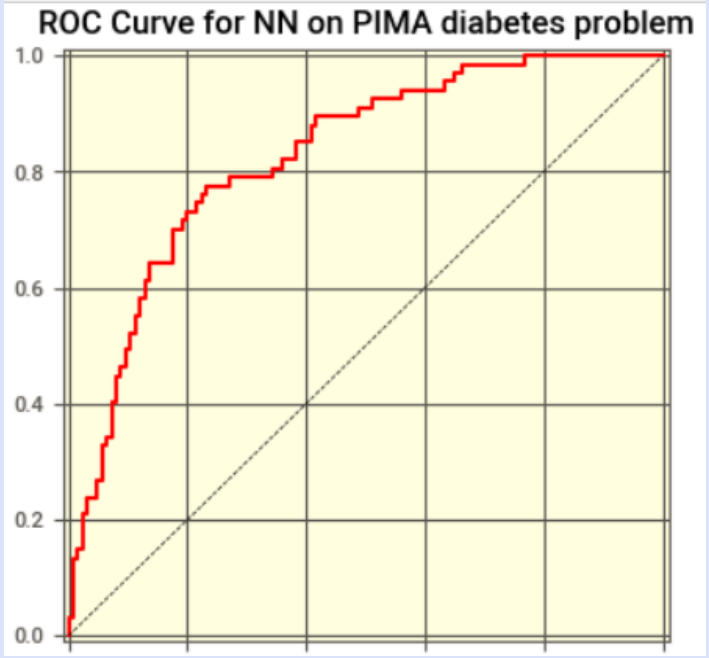
	Features	Importance	Model
1	glucose_tolerance_test	0.175242	SGD
7	age	0.050878	SGD
5	bmi	0.045077	SGD
0	times_pregnant	0.034531	SGD
4	insulin	0.023013	SGD
6	pedigree_function	0.021368	SGD
3	skin_thickness	0.020594	SGD
2	blood_pressure	0.016589	SGD

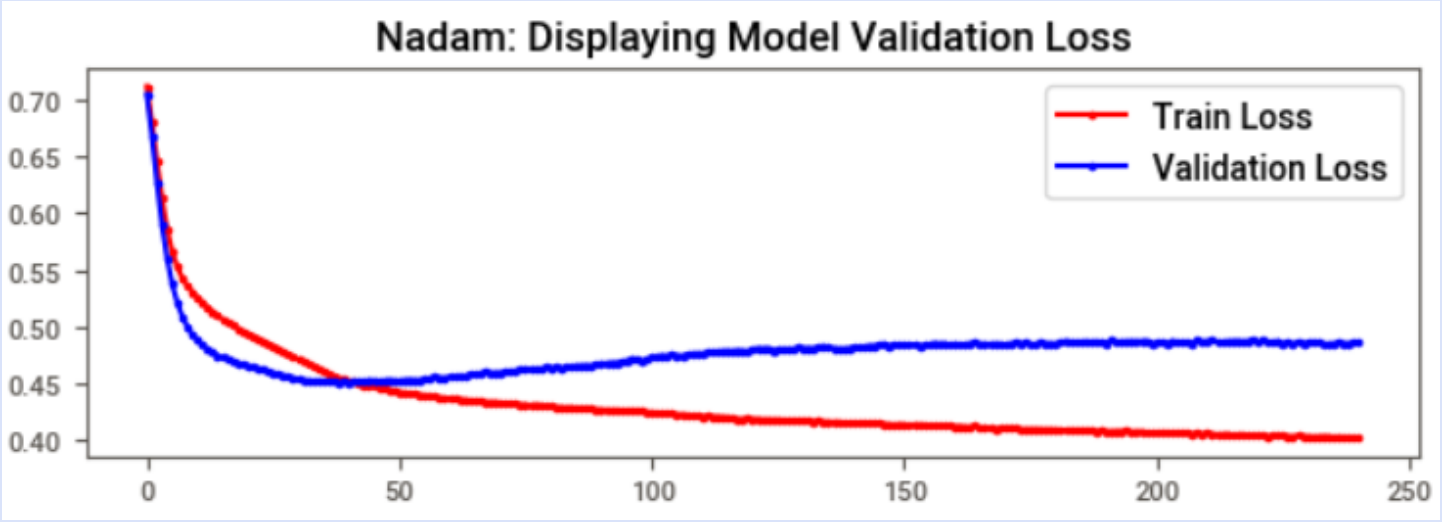
3) Keras Algorithm with Optimizer Nadam:

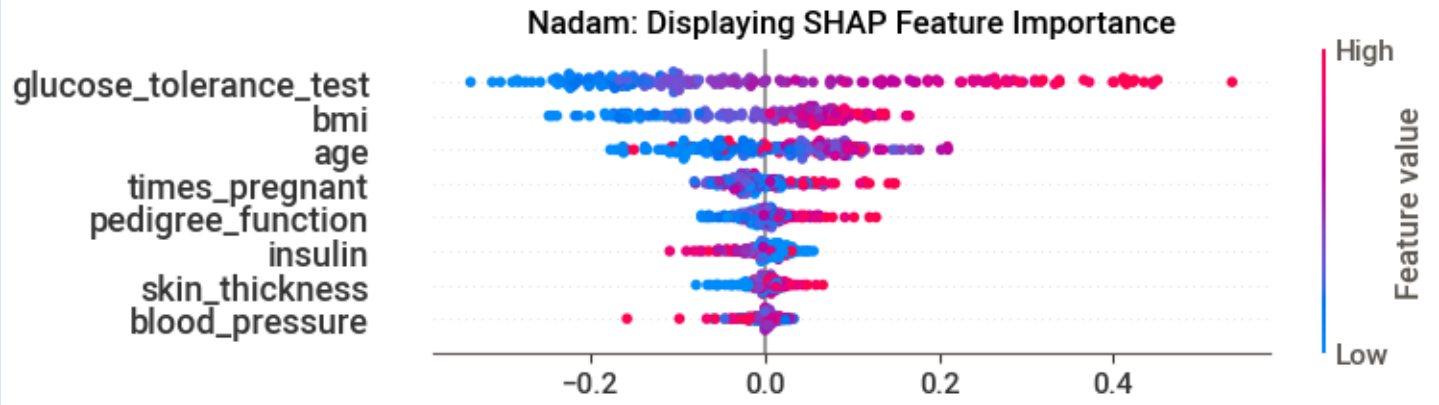
```
Nadam Model with Batch Size: 10 and Epochs: 800
Creating Model
Fitting Model
```

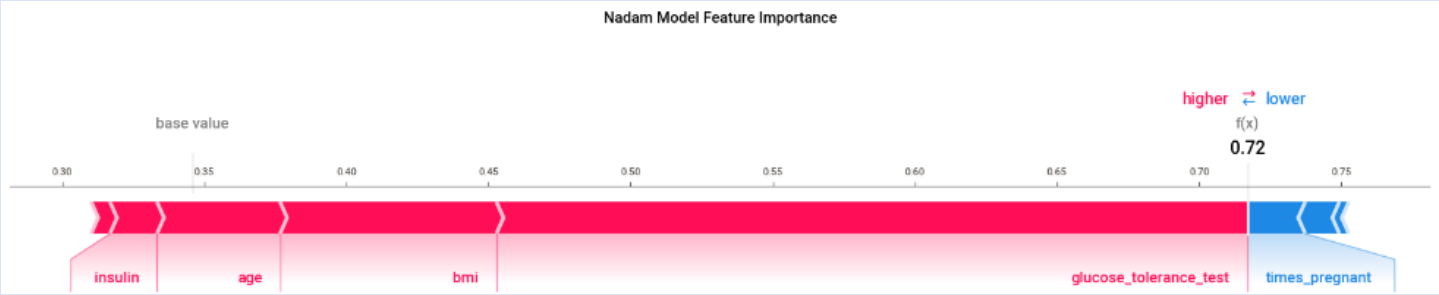
```
Using 576 background data samples could cause slower run times. Co
```

```
Making Model Predictions
Calculating AUC Score
Nadam Model auc_score :0.8262686567164179
Nadam: Computing Model validation loss performance
         loss  accuracy  val_loss  val_accuracy
240  0.402151  0.810764  0.486368      0.776042
  Optimizer  Input Val Loss  Output Val Loss  Input Val Accuracy
0         0        0.000000         0.000000            0.000000
1   RMSprop        0.664082         0.519842            0.697917
2       SGD        0.732052         0.467560            0.572917
3     Nadam        0.704898         0.486368            0.651042
```









Nadam: List Feature Importance

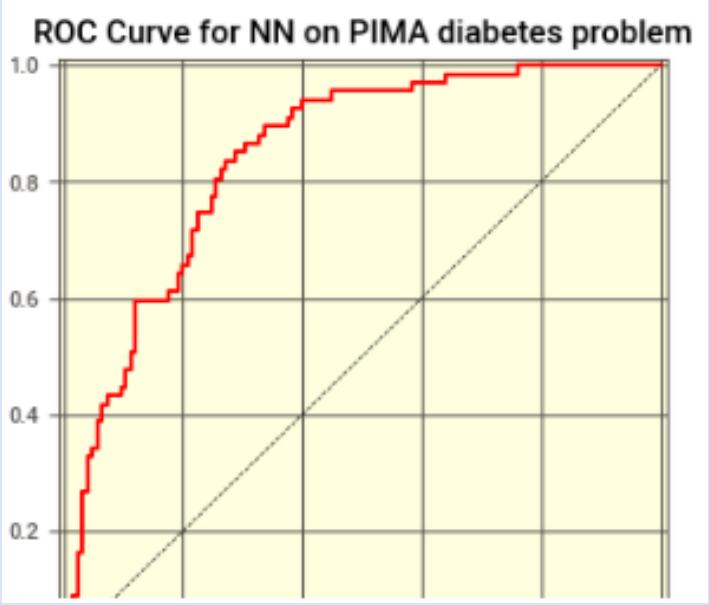
	Features	Importance	Model
1	glucose_tolerance_test	0.179169	Nadam
5	bmi	0.079997	Nadam
7	age	0.071661	Nadam
0	times_pregnant	0.030211	Nadam
6	pedigree_function	0.022472	Nadam
4	insulin	0.017829	Nadam
3	skin_thickness	0.013714	Nadam
2	blood_pressure	0.011544	Nadam

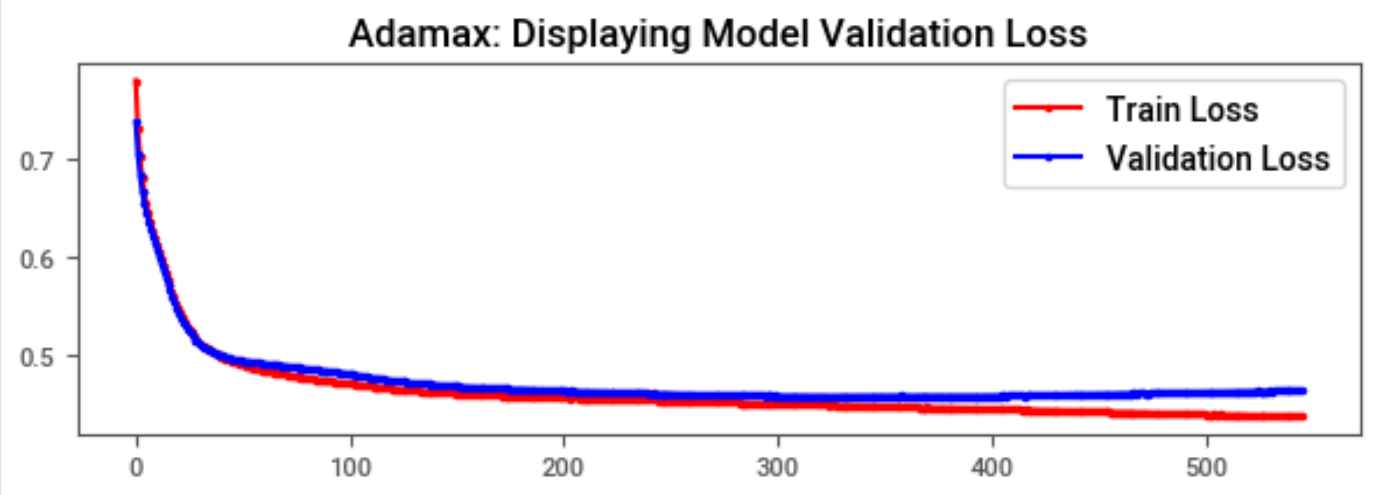
4) Keras Algorithm with Optimizer Adamax:

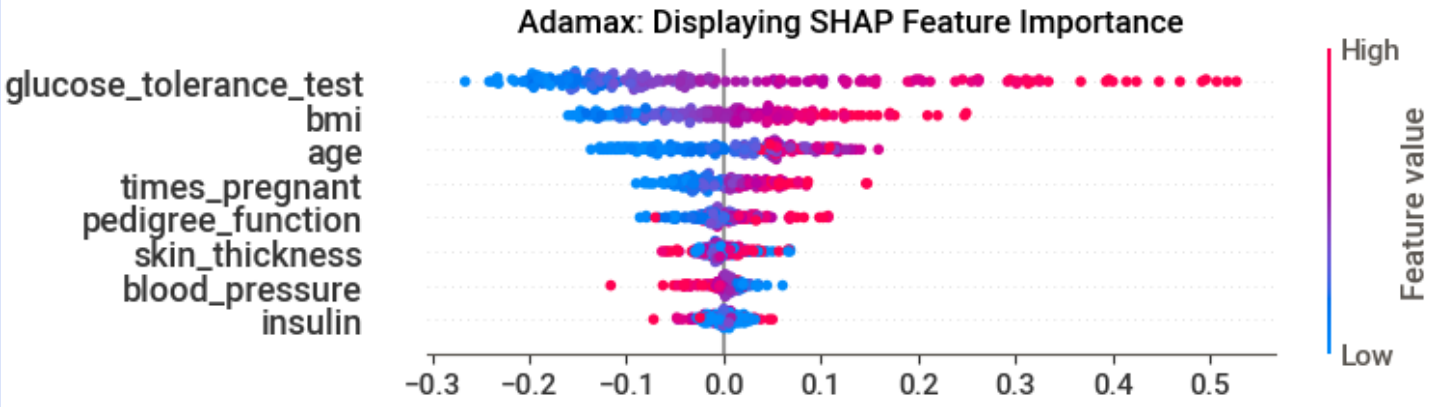
```
Adamax Model with Batch Size: 10 and Epochs: 900
Creating Model
Fitting Model
```

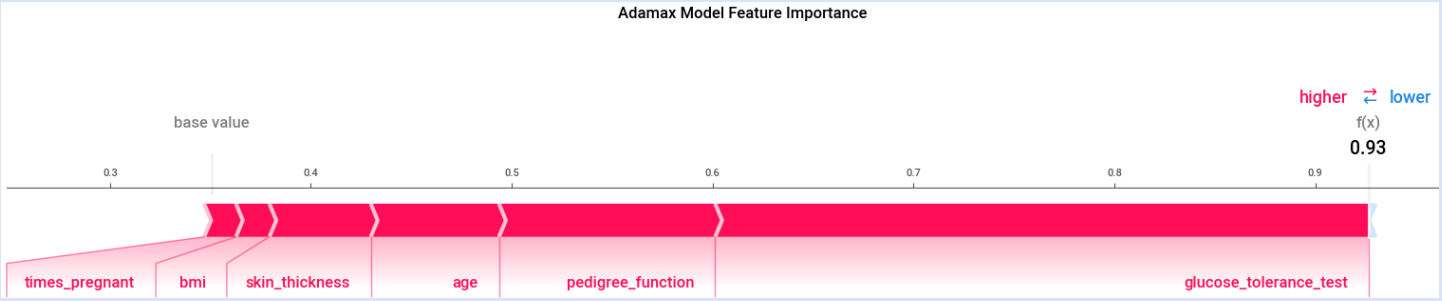
```
Using 576 background data samples could cause slower run times. Consi
```

```
Making Model Predictions
Calculating AUC Score
Adamax Model auc_score :0.8411940298507463
Adamax: Computing Model validation loss performance
         loss  accuracy  val_loss  val_accuracy
545  0.438843  0.788194   0.46459      0.776042
  Optimizer  Input Val Loss  Output Val Loss  Input Val Accuracy  \
0         0        0.000000         0.000000            0.000000
1   RMSprop        0.664082         0.519842            0.697917
2       SGD        0.732052         0.467560            0.572917
3     Nadam        0.704898         0.486368            0.651042
4    Adamax        0.736525         0.464590            0.432292
```







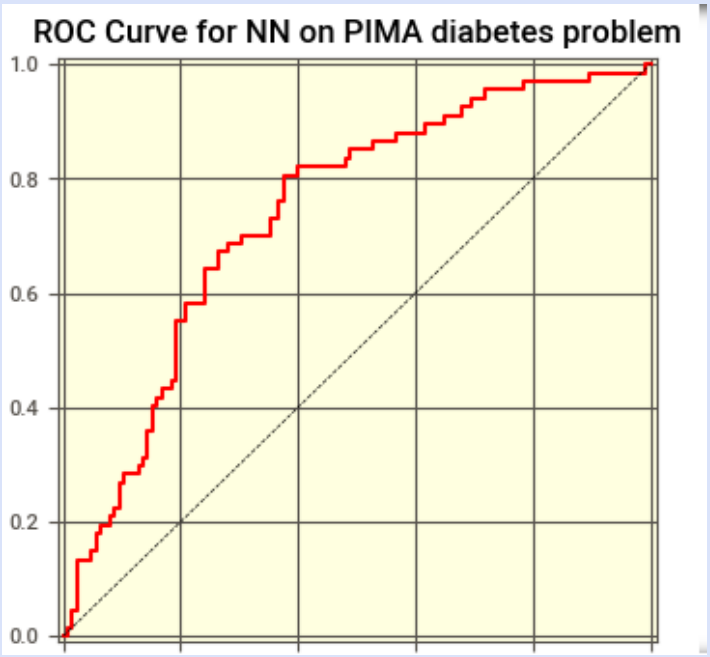


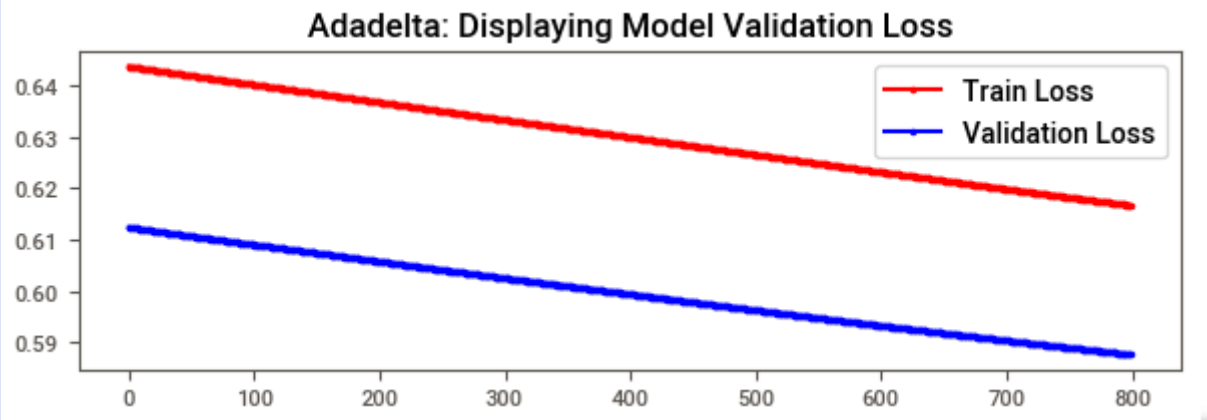
Adamax: List Feature Importance

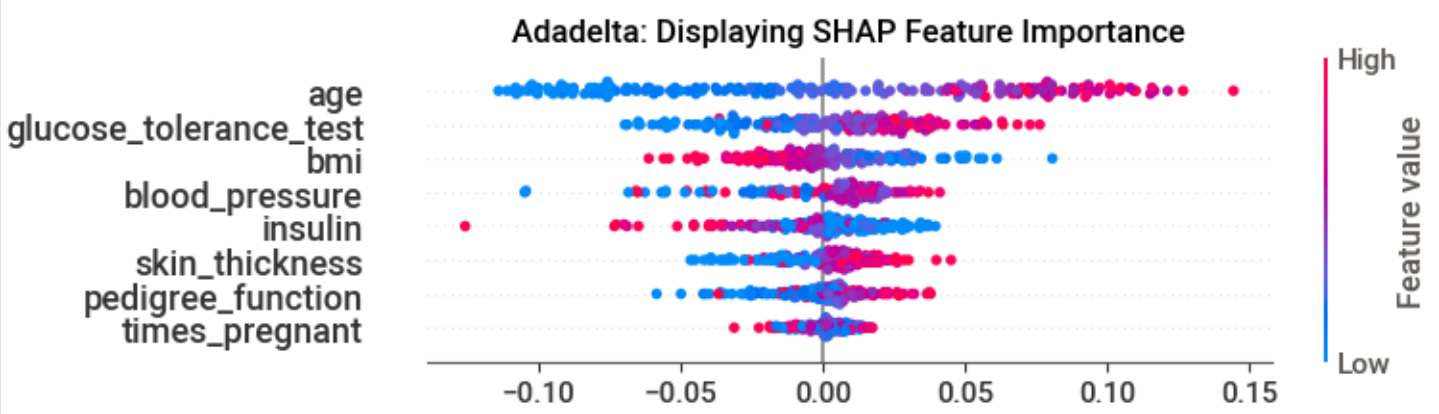
	Features	Importance	Model
1	glucose_tolerance_test	0.158482	Adamax
5	bmi	0.069657	Adamax
7	age	0.061823	Adamax
0	times_pregnant	0.032609	Adamax
6	pedigree_function	0.023062	Adamax
3	skin_thickness	0.015775	Adamax
2	blood_pressure	0.013057	Adamax
4	insulin	0.012496	Adamax

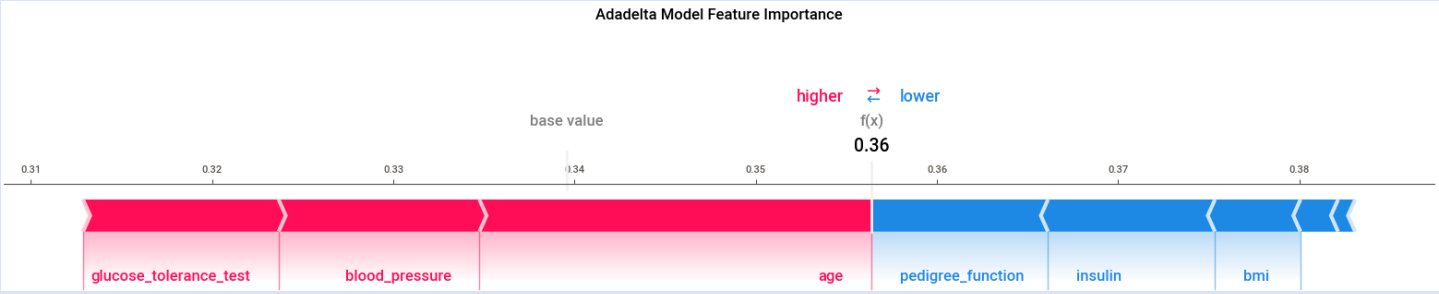
5) Keras Algorithm with Optimizer Adadelta:

```
Adadelta Model with Batch Size: 20 and Epochs: 800
Creating Model
Fitting Model

Using 576 background data samples could cause slower run times. Consi

Making Model Predictions
Calculating AUC Score
Adadelta Model auc_score :0.7383880597014926
Adadelta: Computing Model validation loss performance
         loss  accuracy  val_loss  val_accuracy
799  0.616719  0.651042  0.587543      0.651042
  Optimizer  Input Val Loss  Output Val Loss  Input Val Accuracy  \
0         0        0.000000         0.000000            0.000000
1   RMSprop        0.664082         0.519842            0.697917
2       SGD        0.732052         0.467560            0.572917
3     Nadam        0.704898         0.486368            0.651042
4    Adamax        0.736525         0.464590            0.432292
5  Adadelta        0.612426         0.587543            0.651042
```







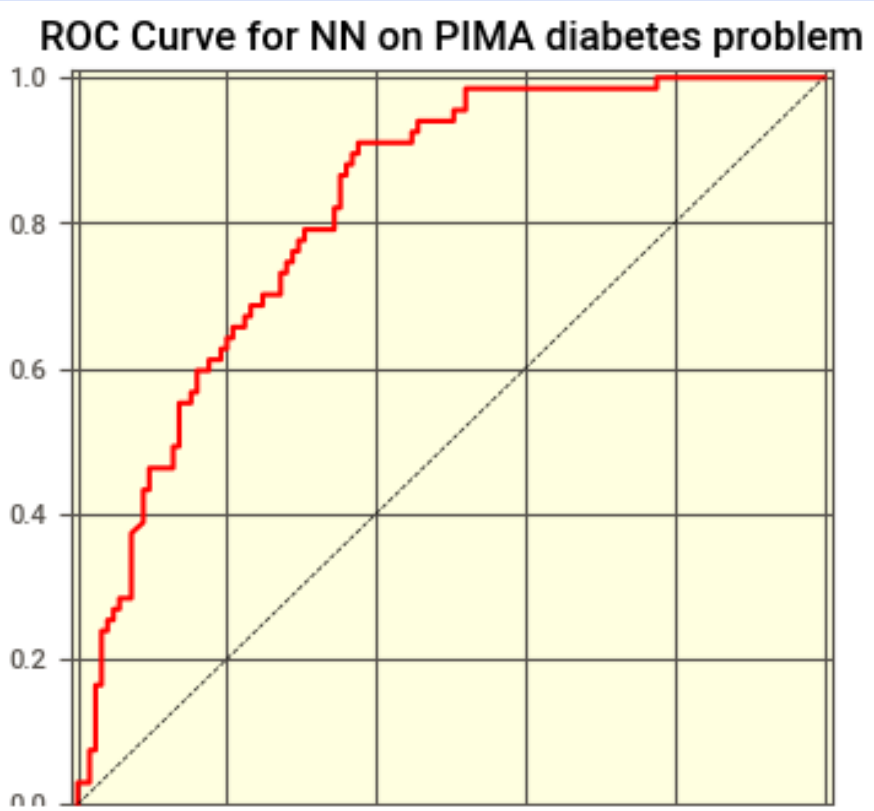


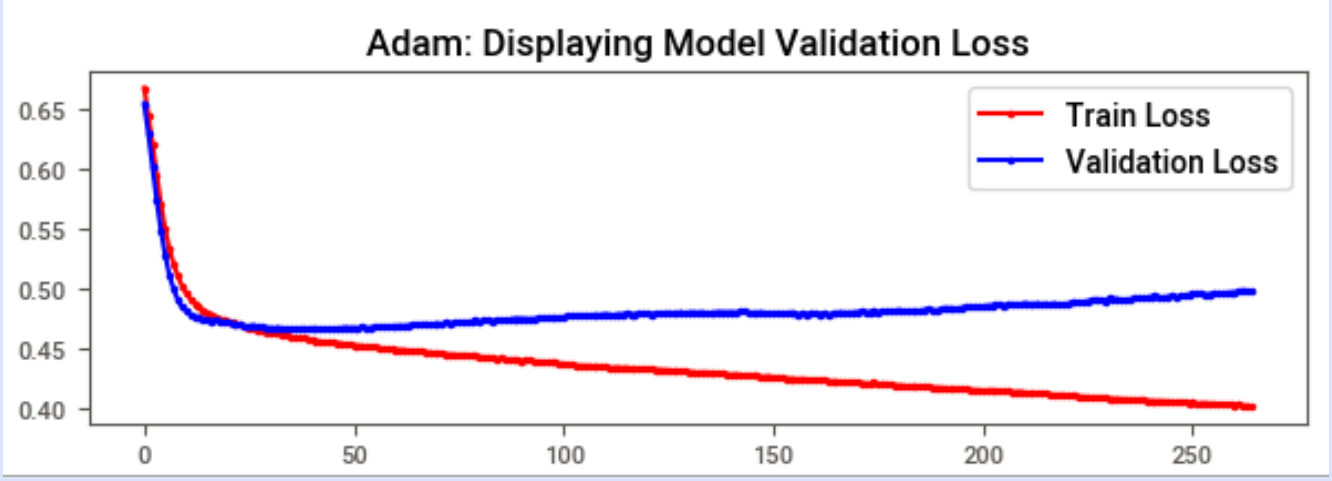
Adadelta: List Feature Importance

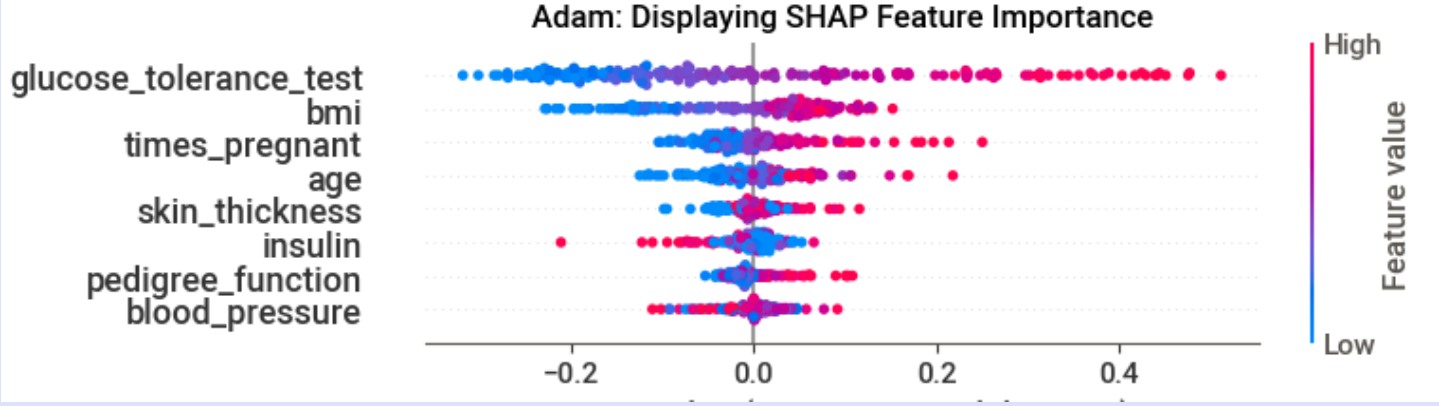
	Features	Importance	Model
7	age	0.060745	Adadelta
1	glucose_tolerance_test	0.025040	Adadelta
5	bmi	0.018069	Adadelta
2	blood_pressure	0.016655	Adadelta
4	insulin	0.016061	Adadelta
3	skin_thickness	0.012444	Adadelta
6	pedigree_function	0.011359	Adadelta
0	times_pregnant	0.006161	Adadelta

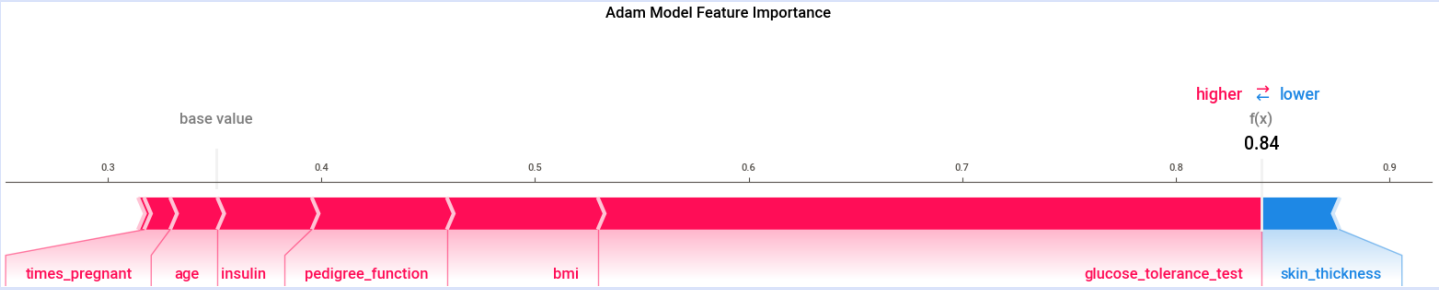
6) Keras Algorithm with Optimizer Adam:

```
Adam Model auc_score :0.8208955223880596
Adam: Computing Model validation loss performance
         loss  accuracy  val_loss  val_accuracy
264  0.401989  0.815972  0.498047      0.744792
  Optimizer  Input Val Loss  Output Val Loss  Input Val Accuracy  \
0         0        0.000000         0.000000            0.000000
1   RMSprop        0.664082         0.519842            0.697917
2       SGD        0.732052         0.467560            0.572917
3     Nadam        0.704898         0.486368            0.651042
4    Adamax        0.736525         0.464590            0.432292
5  Adadelta        0.612426         0.587543            0.651042
6      Adam        0.655232         0.498047            0.671875
```







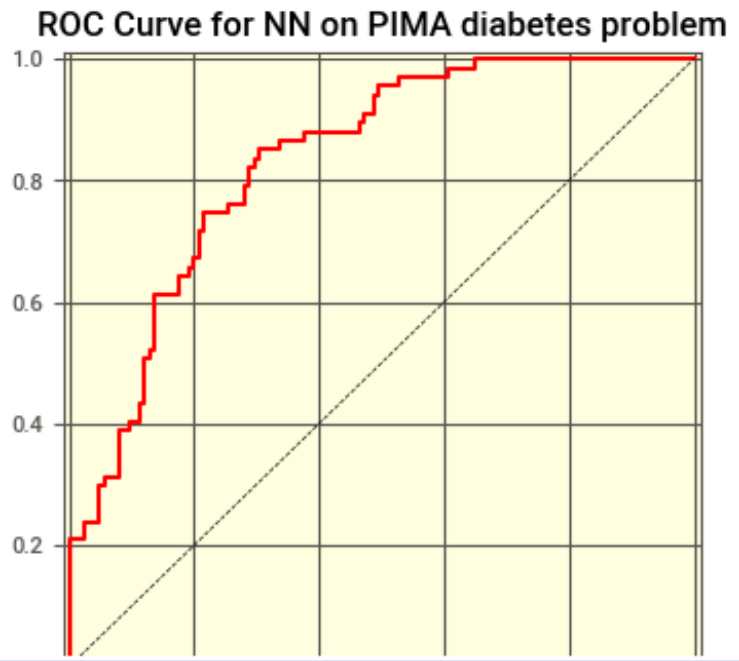


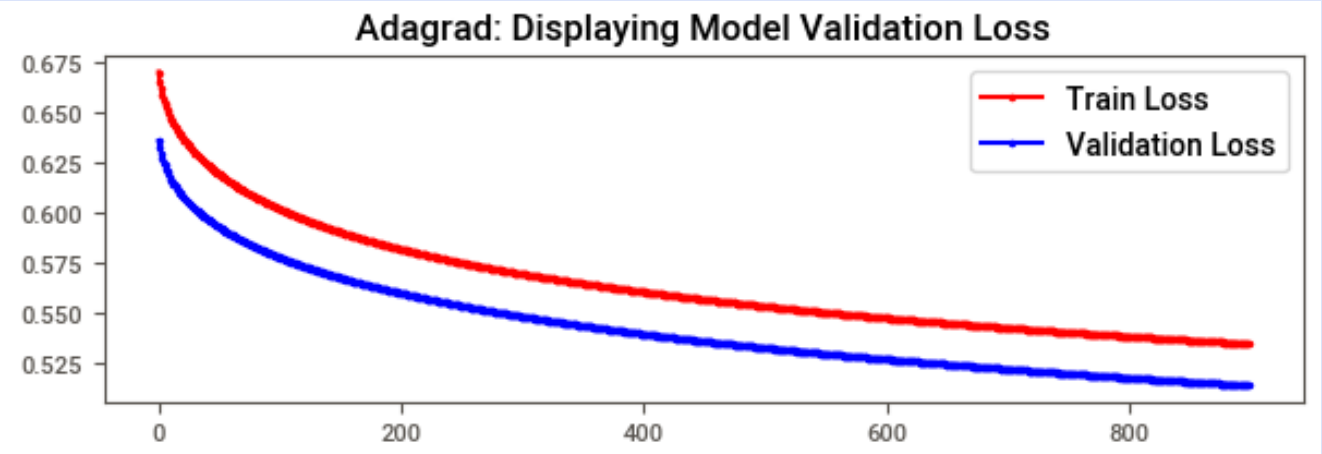
Adam: List Feature Importance

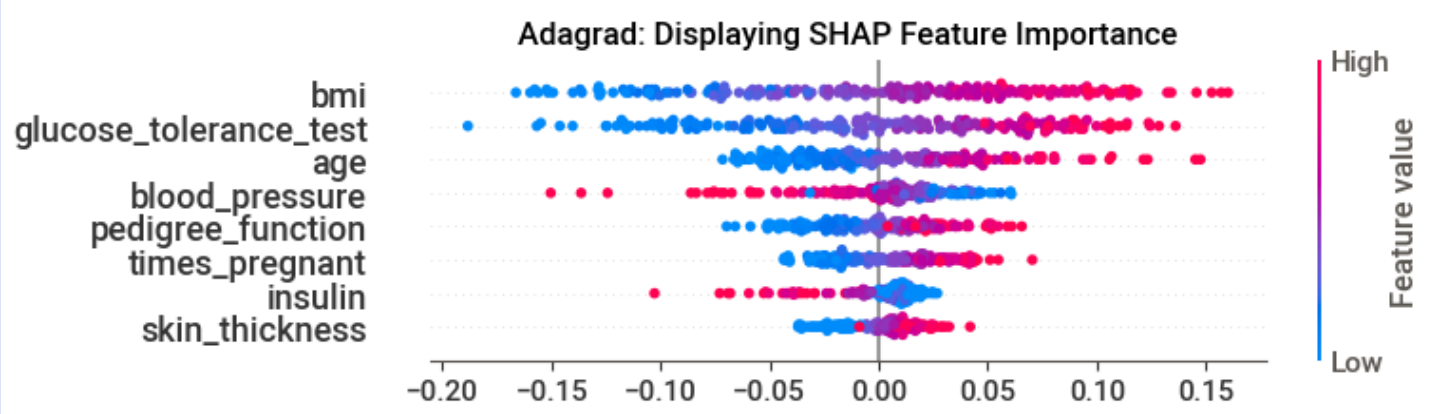
	Features	Importance	Model
1	glucose_tolerance_test	0.170184	Adam
5	bmi	0.072324	Adam
0	times_pregnant	0.040196	Adam
7	age	0.035781	Adam
3	skin_thickness	0.020557	Adam
4	insulin	0.019800	Adam
6	pedigree_function	0.019177	Adam
2	blood_pressure	0.018084	Adam

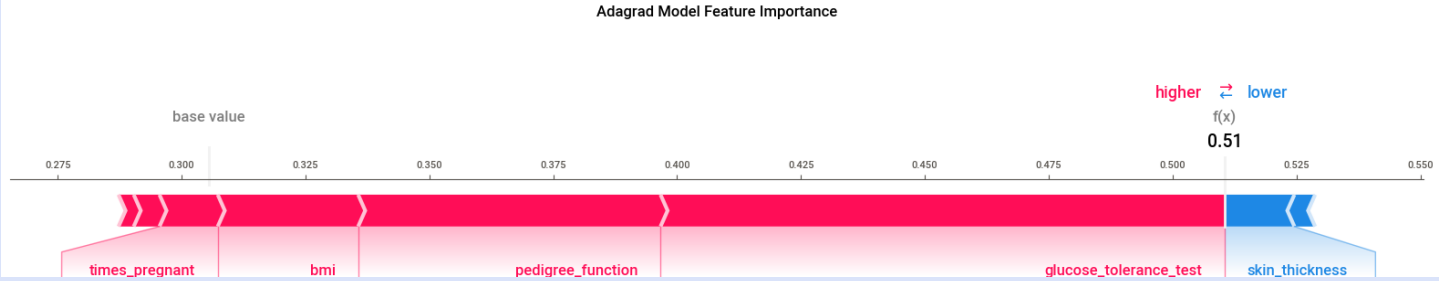
7) Keras Algorithm with Optimizer Adagrad:

```
Making Model Predictions
Calculating AUC Score
Adagrad Model auc_score :0.8320000000000001
Adagrad: Computing Model validation loss performance
         loss  accuracy  val_loss  val_accuracy
899  0.534858  0.682292  0.514088       0.71875
  Optimizer  Input Val Loss  Output Val Loss  Input Val Accuracy  \
0         0        0.000000         0.000000            0.000000
1   RMSprop        0.664082         0.519842            0.697917
2       SGD        0.732052         0.467560            0.572917
3     Nadam        0.704898         0.486368            0.651042
4    Adamax        0.736525         0.464590            0.432292
5  Adadelta        0.612426         0.587543            0.651042
6      Adam        0.655232         0.498047            0.671875
7   Adagrad        0.635958         0.514088            0.645833
```









Adagrad: List Feature Importance

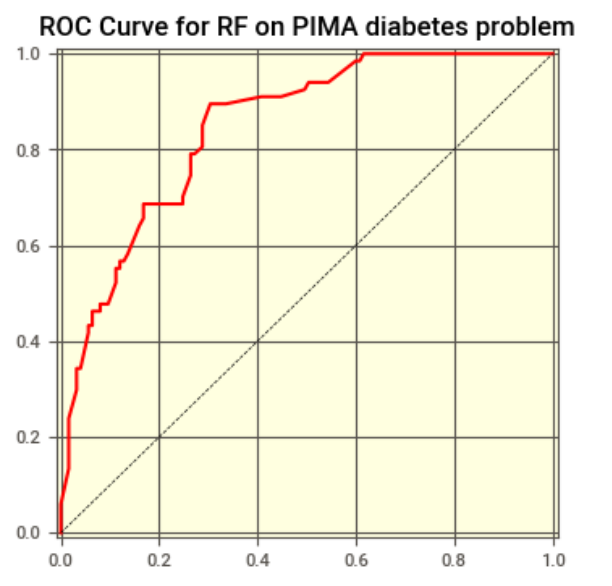
	Features	Importance	Model
5	bmi	0.064633	Adagrad
1	glucose_tolerance_test	0.059301	Adagrad
7	age	0.036672	Adagrad
2	blood_pressure	0.023516	Adagrad
6	pedigree_function	0.022678	Adagrad
0	times_pregnant	0.021181	Adagrad
4	insulin	0.013991	Adagrad
3	skin_thickness	0.013115	Adagrad

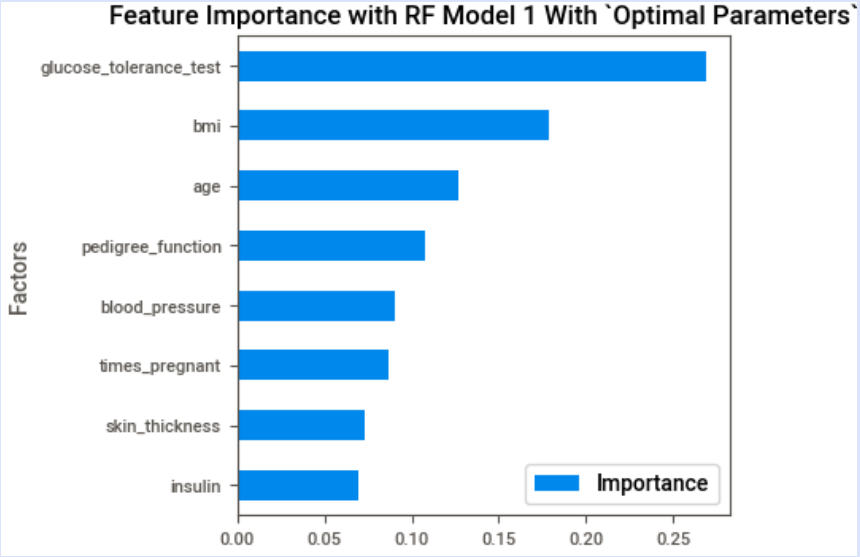
8) Base Model: Random Forest:

RF Model WITH Auto-Hypertuned Parameters

```
#Running the Model...Call Method to 'Build Simple Random Forest With Class Weights'
op_preds, op_model = build_op_rf(X_train, y_train, X_test, best_params=optimal_rf_params) # Call Method to 'Build Simple Random Forest Without Class Weights'

#Collect Results
original_results = evaluate(y_test, op_preds, eval_type="RForest")
original_results["Factors"] = 'RF1_features_df'
rf_acc = original_results['auc'] # Extract Roc_acc for comparison with deep learning models
print(original_results, "\nRF_acc: ",rf_acc)

# Get Factors Contributing to Diabetes
RF1_features_df = feature_importance(X, op_model)[:40]
RF1_features_df = RF1_features_df[RF1_features_df['Importance'] > 0] # Filter Features with Negative Correlation
RF1_features_df = RF1_features_df.sort_values(by=['Importance'],ascending=True) # Attach Importance to data before split

# Capture Model Drivers in  Dataframe
RF1_drivers = [['Uses train test set to achieve best hypertuninge to resample data from train set to achieve balanced class.'],
               ['No optimal parameters were used.'],
               ['Runs random forest model with above drivers to get top contributory features.']] # initialize list of lists
RF1_drivers = pd.DataFrame(RF1_drivers, columns=['Main Drivers behind RF1 model are as follows:']) # Create the pandas DataFrame
RF1_drivers.index = np.arange(1,len(RF1_drivers)+1)

# Plot Features
RF1_features_df.set_index('Factors', inplace=True)
RF1_features_df.plot(kind='barh', figsize=(4, 4))
plt.title('Feature Importance with RF Model 1 With `Optimal Parameters`')

# Merge Description to Encoded Features/Factors
RF1_features_df = RF1_features_df.rename_axis('Factors').reset_index() # Reset index before copying it to column
#RF1_features_df = pd.merge(RF1_features_df,factor_list,on='Factors',how='left',indicator=True).replace(np.nan, "") # Merge Data
#RF1_features_df['Description'] = np.where(RF1_features_df['Description'] == '/', RF1_features_df['Factors'], RF1_features_df['Description'])
#RF1_features_df['Description'] = RF1_features_df['Description'].replace(r'_', ' ', regex=True) # Replace Underscore with single space
final_columns = ['Factors', 'Importance', 'Description'] # List of Columns to keep
RF1_features_df = RF1_features_df.drop(columns=[col for col in RF1_features_df if col not in final_columns]) # Drop Columns if not in list
#print(op_preds)
```

```
{'type': 'RForest', 'accuracy': 0.7760416666666666, 'recall': 0.6865671641791045, 'precision': 0.6764705882352942, 'fscore': 0.6861732644865174, 'auc': 0.7552835820895524, 'Factors': 'RF1_features_df'}
RF_acc:  0.7552835820895524
```





3d) Model Choice

3d-i) Result Summary

Although, the assignment required construction of Deep Learning Models (DLM), however, Random Forest was also included as a baseline measure to compare performance of DLM against other unsupervised classification algorithms. As evident from the Result Summary Table below, based on ROC AUC Score, all variations of DLM outperformed Random Forest Model, indicating it to be a better choice for diabetes prediction.

Result Summary Table:

	Optimizer	Input Val Loss	Output Val Loss	Input Val Accuracy	Output Val Accuracy	AUC Score	Epochs	Actual Epochs	val_loss_diff	val_acc_diff	per_epochs_diff	epochs_diff
0	Adagrad	0.636	0.514	0.646	0.719	0.832	900	900	-0.122	0.073	100	0
1	RF	0.000	0.000	0.000	0.000	0.755	0	0	0.000	0.000	0	0
2	Adadelta	0.612	0.588	0.651	0.651	0.738	800	800	-0.025	0.000	100	0
3	Adamax	0.737	0.465	0.432	0.776	0.841	900	546	-0.272	0.344	60	354
4	SGD	0.732	0.468	0.573	0.766	0.851	800	394	-0.264	0.193	49	406
5	Nadam	0.705	0.486	0.651	0.776	0.826	800	241	-0.219	0.125	30	559
6	Adam	0.655	0.498	0.672	0.745	0.821	900	265	-0.157	0.073	29	635
7	RMSprop	0.664	0.520	0.698	0.745	0.811	900	265	-0.144	0.047	29	635

3d-ii) Model Ranking, Choice and Justification

	Optimizer	Input Val Loss	Output Val Loss	Input Val Accuracy	Output Val Accuracy	AUC Score	Epochs	Actual Epochs	val_loss_diff	val_acc_diff	per_epochs_diff	epochs_diff	Rank
0	Adagrad	0.636	0.514	0.646	0.719	0.832	900	900	-0.122	0.073	100	0	1
1	SGD	0.732	0.468	0.573	0.766	0.851	800	394	-0.264	0.193	49	406	2
2	Adamax	0.737	0.465	0.432	0.776	0.841	900	546	-0.272	0.344	60	354	2
3	Nadam	0.705	0.486	0.651	0.776	0.826	800	241	-0.219	0.125	30	559	2
4	Adam	0.655	0.498	0.672	0.745	0.821	900	265	-0.157	0.073	29	635	2
5	RMSprop	0.664	0.520	0.698	0.745	0.811	900	265	-0.144	0.047	29	635	2
6	RF	0.000	0.000	0.000	0.000	0.755	0	0	0.000	0.000	0	0	3
7	Adadelta	0.612	0.588	0.651	0.651	0.738	800	800	-0.025	0.000	100	0	4

Recommended Model and Justification:

Since performance measurement is an essential task in machine learning,
hence, for classification problems, AUC-ROC Curve is a reliable measure.
As such, for visualizing performance of a multi-class diabetes prediction problem,
the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve has been utilized since it is one of the most important evaluation metrics for checking any classification model's performance.

Apart from AUC Score, validation loss was also taken into account which allowed final model selection on the premise that validation loss continuously decined to ensure no overfitting has occured.

Based on the above criterion, Best Model was found to be Adagrad with an roc_auc of 0.832.
which successfully satisfied both primary and secondary objectives.

Hence, we recommend this model for Diabetes Prediction.

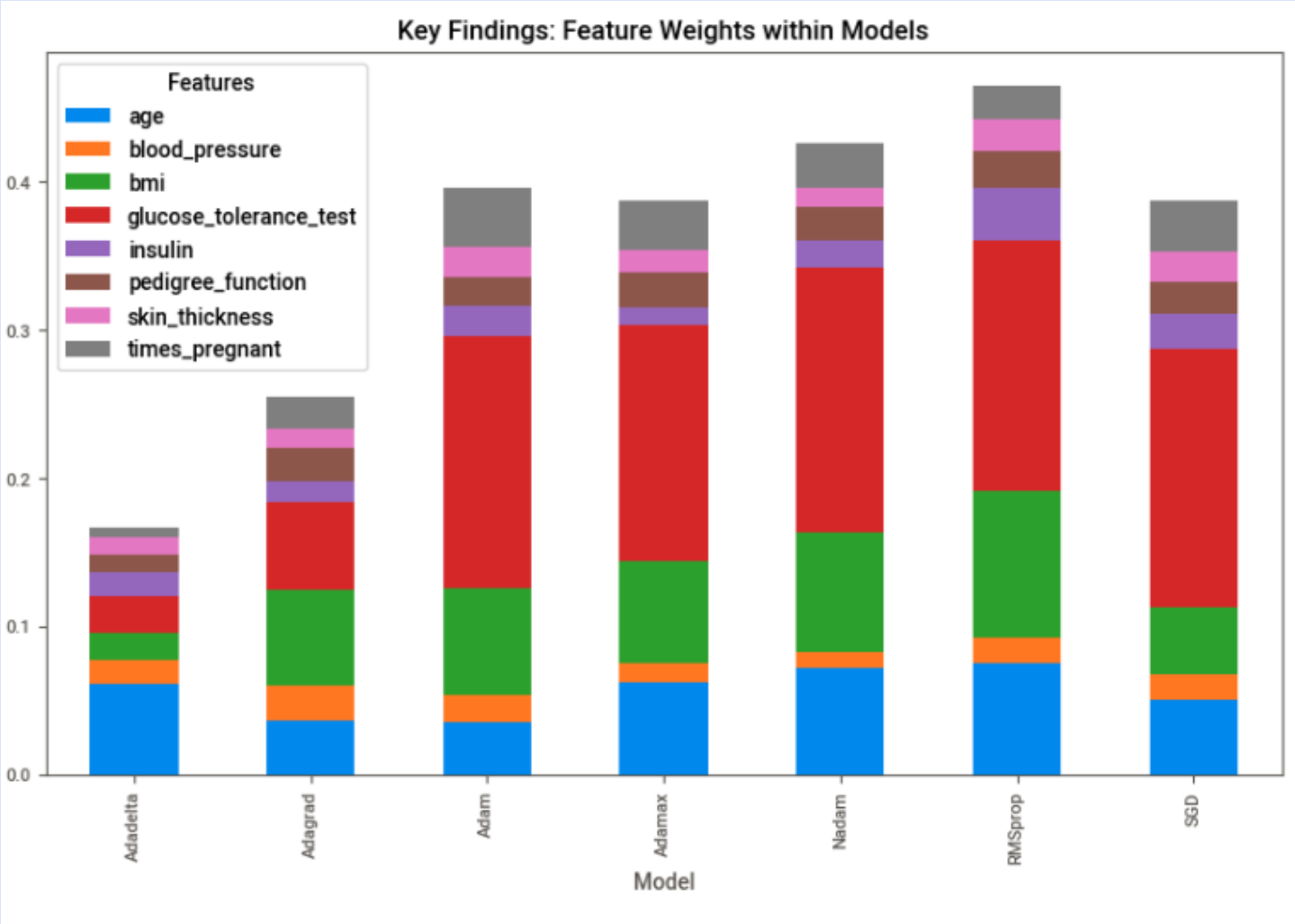
Adagrad Model Features in terms of importance are given below:

	Features	Importance
0	bmi	0.064633
1	glucose_tolerance_test	0.059301
2	age	0.036672
3	blood_pressure	0.023516
4	pedigree_function	0.022678
5	times_pregnant	0.021181
6	insulin	0.013991
7	skin_thickness	0.013115

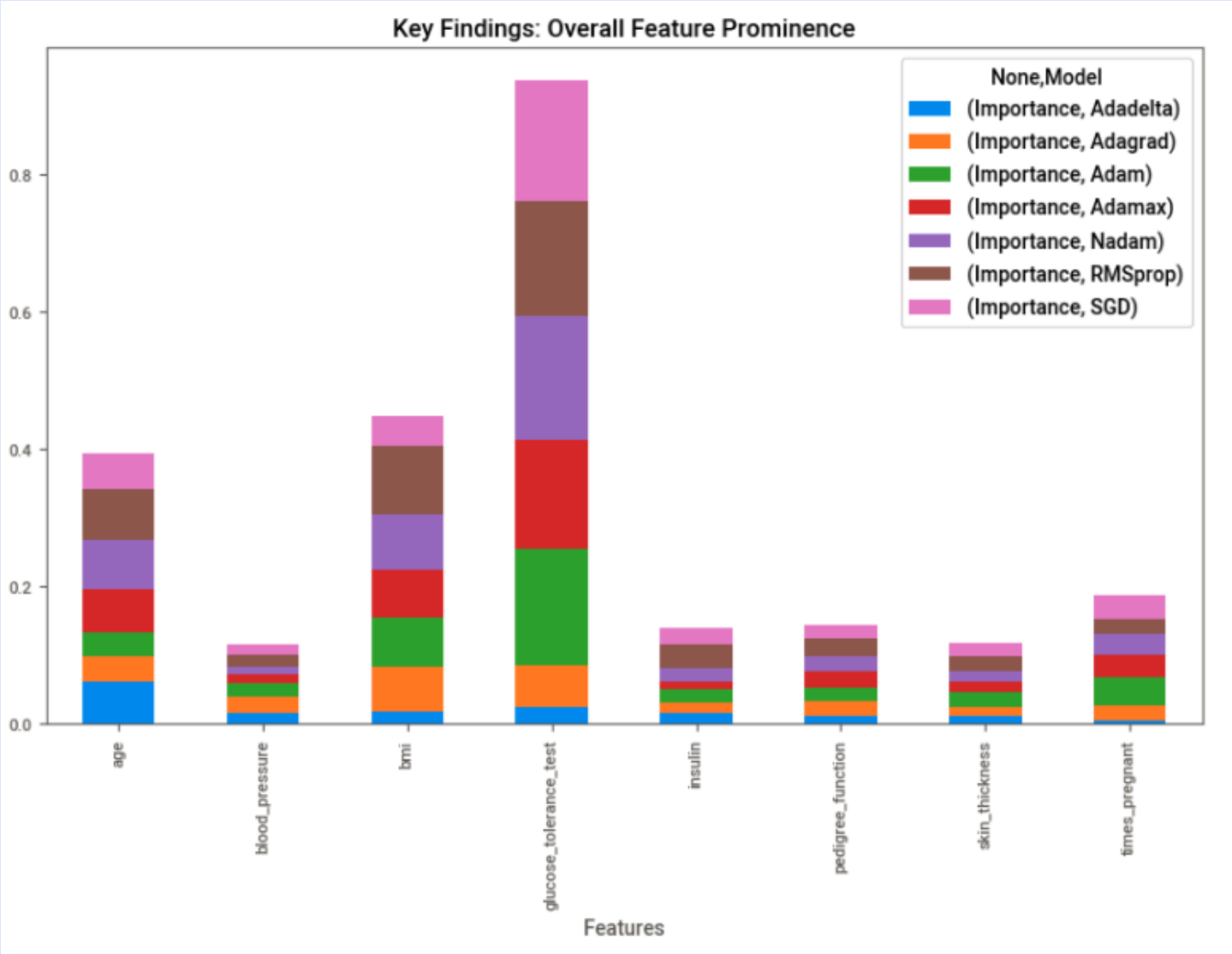
4) Summary Key Findings and Insights

Model output clearly shows glucose tolerance test, bmi and age as most prominent features in most models irrespective of operator variation or auc score. Consequently, feature weights and their individual prominence is depicted as follows:

4a) Feature Weights within Models



4b Overall Feature Prominence



5) Model Shortcomings and Future Directions

5a) Model Shortcomings

Despite its novel approach to automatically pick best model with best parameters, the project is not without its shortcomings.

To begin with, it only utilized a very small dataset with merely nine features.

Hence, it may have discounted other important factors that may impact on set of diabetes.

Furthermore, the project could have employed other deep learning models to ensure even more improved results.

Then, employed Grid Search Hyperparameter Tuning proved to be computationally intensive and could have been replaced by more efficient methods. This also led to a basic model architecture which discounted incorporating other parameters like different learning rates, momentum, etc.

5b) Plan of Action to Revisit Analysis

Future recommendations are, hence, as follows:

1) Use a larger dataset with more features.

2) Employ more unsupervised models like LSTM etc.

3) Try other Hyperparameter Tuning methods like Keras Tuner or Bayesian optimization (See, Section 5).

4) Include additional parameter search like learning rate, momentum, neuron activation function, etc. to make a more robust model.

6 Useful Links

6a) Guide to Keras

https://machinelearningmastery.com/use-keras-deep-learning-models-scikit-learn-python/

6b) Keras Hyperparameter Tuning Approaches

https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

https://medium.datadriveninvestor.com/hyperparameter-tuning-with-deep-learning-grid-search-8630aa45b2da

https://towardsai.net/p/l/stop-using-grid-search-the-complete-practical-tutorial-on-keras-tuner

6c) Model Checkpoints

https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

https://pyimagesearch.com/2021/06/30/how-to-use-the-modelcheckpoint-callback-with-keras-and-tensorflow/

6d) Get Feature Importance in Keras

https://stackoverflow.com/questions/45361559/feature-importance-chart-in-neural-network-using-keras-in-python

https://www.kdnuggets.com/2020/01/explaining-black-box-models-ensemble-deep-learning-lime-shap.html

6e) Shap Plots: Some Good Examples

https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/decision_plot.html

6f) Matplot Lib

https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplot.html

https://medium.com/mlearning-ai/shap-force-plots-for-classification-d30be430e195

6f) Other Useful Models

https://github.com/AI-MOO/IBM-Machine-Learning-Professional-Certificate

https://github.com/topics/ibm-machine-learning

https://samyzaf.com/ML/pima/pima.html

7) Github Links

7a) Link to Main Folder

https://github.com/FATIMASP/IBM-MACHINE-LEARNING-CERTIFICATION/tree/main/Deep%20Learning%20and%20Reinforcement%20Learning

7b) Link to Assignment Notebook

https://github.com/FATIMASP/IBM-MACHINE-LEARNING-CERTIFICATION/blob/main/Deep%20Learning%20and%20Reinforcement%20Learning/DEEP%20LEARNING%20%26%20REINFORCEMENT%20LEARNING%20FINAL%20MODEL.ipynb

8) References

Naz, H. & Ahuja, S., 2020. Deep learning approach for diabetes prediction using PIMA Indian dataset. *Journal of Diabetes and Metabolic Disorders*, 19(1), p. 391–403.

The grading will center around 5 main points:

1. **Does the report include a section describing the data?**
2. **Does the report include a paragraph detailing the main objective(s) of this analysis?**
3. **Does the report include a section with variations of a Deep Learning model and specifies which one is the model that best suits the main objective(s) of this analysis?**
4. **Does the report include a clear and well presented section with key findings related to the main objective(s) of the analysis?**
5. **Does the report highlight possible flaws in the model and a plan of action to revisit this analysis with additional data or different modeling techniques?**