Extracted Code from Notebook

Code Cell

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
import numpy as np
import pandas as pd
import os

# Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load dataset
file_path = "C:/Users/CHUMKI/Desktop/AA/Diabetes.xlsx"
df = pd.read_excel(file_path,usecols=None)

# Drop unnecessary columns
df.drop(columns=["encounter_id", "patient_nbr", "max_glu_serum",
"A1Cresult","examide","citoglipton"], inplace=True)
```

Code Cell

```
# Encode categorical variables
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
   df[col] = LabelEncoder().fit_transform(df[col].astype(str))
```

Code Cell

```
# Define features and target variable
X = df.drop(columns=['Readmission'])
y = df['Readmission']

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

# Apply Standard Scaling
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Preprocessing Complete!")
```

Code Cell

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

Code Cell

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

# Dictionary to store results
results = {}
```

Code Cell

```
# Helper function to evaluate models
def evaluate_model(model, name):
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else
None

    results[name] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1-Score": f1_score(y_test, y_pred),
        "AUROC": roc_auc_score(y_test, y_pred_prob) if y_pred_prob is not None else "N/A"
    }
```

Code Cell

```
# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
evaluate_model(rf, "Random Forest")
```

Code Cell

```
# XGBoost
xgb = XGBClassifier(eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)
evaluate_model(xgb, "XGBoost")
```

Code Cell

```
# SVM
#svm = SVC(kernel="rbf", probability=True, max_iter=5000, random_state=42)
#svm.fit(X_train, y_train)
#evaluate_model(svm, "SVM")
```

Code Cell

```
# Logistic Regression
lr = LogisticRegression(max_iter=5000, random_state=42)
lr.fit(X_train, y_train)
evaluate_model(lr, "Logistic Regression")
```

Code Cell

```
# LDA
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
evaluate_model(lda, "LDA")
```

Code Cell

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input

# LSTM Model
X_train_lstm = np.array(X_train).reshape(X_train.shape[0], X_train.shape[1], 1)
X_test_lstm = np.array(X_test).reshape(X_test.shape[0], X_test.shape[1], 1)
model = Sequential([
  Input(shape=(X_train.shape[1],)),  # ✅ Explicit Input layer
  Dense(64, activation='relu'),
  Dense(32, activation='relu'),
  Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)

# Evaluate LSTM
y_pred_lstm = (model.predict(X_test) > 0.5).astype(int).flatten()
y_pred_prob_lstm = model.predict(X_test).flatten()

results["LSTM"] = {
   "Accuracy": accuracy_score(y_test, y_pred_lstm),
   "Precision": precision_score(y_test, y_pred_lstm),
   "Recall": recall_score(y_test, y_pred_lstm),
   "F1-Score": f1_score(y_test, y_pred_lstm),
   "AUROC": roc_auc_score(y_test, y_pred_prob_lstm)
}
```

Code Cell

```
results_df = pd.DataFrame(results).T
print(results_df)
```

Code Cell

```
import numpy as np
import pandas as pd
import shap
from sklearn.inspection import permutation_importance

# Store feature importance
feature_importance = pd.DataFrame(index=X_train.columns)

# Random Forest Importance
feature_importance["Random Forest"] = rf.feature_importances_

# XGBoost Importance
feature_importance["XGBoost"] = xgb.feature_importances_

# Logistic Regression Importance (absolute coefficients)
feature_importance["Logistic Regression"] = np.abs(lr.coef_[0])

# LDA Importance (absolute coefficients)
feature_importance["LDA"] = np.abs(lda.coef_[0])

# SVM Feature Importance (only if linear kernel is used)
#if hasattr(svm, "coef_"):
   #feature_importance["SVM"] = np.abs(svm.coef_[0])
#LSTM
```

```
from scikeras.wrappers import KerasClassifier

class KerasModelWrapper:
    def __init__(self, model):
        self.model = model  # Store the Keras model

    def fit(self, X, y, **kwargs):
        """Fit the model using the provided training data."""
        return self.model.fit(X, y, **kwargs)

    def predict(self, X):
        """Generate predictions from the model and return binary outputs."""
        return (self.model.predict(X) > 0.5).astype(int)

    def score(self, X, y):
        """Return accuracy score for scikit-learn compatibility."""
        from sklearn.metrics import accuracy_score
        return accuracy_score(y, self.predict(X))

# Wrap your model
wrapped_model = KerasModelWrapper(model)

# Ensure y_test is binary
y_test_binary = (y_test > 0.5).astype(int)

# Now compute permutation importance
perm_importance = permutation_importance(wrapped_model, X_test, y_test_binary,
scoring='accuracy', random_state=42)

# Store the results
feature_importance["LSTM"] = perm_importance.importances_mean


# Compute average importance across models
feature_importance["Mean Importance"] = feature_importance.mean(axis=1)

# Sort by overall importance
feature_importance = feature_importance.sort_values(by="Mean Importance",
ascending=False)

# Display top 20 important variables
print(feature_importance.head(20))
```

Code Cell

```
import matplotlib.pyplot as plt
feature_importance["Mean Importance"].head(20).plot(kind="barh", figsize=(10, 6))
plt.gca().invert_yaxis()
plt.xlabel("Mean Importance")
plt.ylabel("Feature Name")
plt.title("Top 20 Most Important Features Across Models")
plt.show()
```

Code Cell

```
import nbformat
from docx import Document

# Load the Jupyter Notebook
notebook_path = "Untitled7.ipynb"  # Change to your notebook's filename
word_output_path = "modcode.docx"

# Read the notebook
with open(notebook_path, "r", encoding="utf-8") as f:
    nb = nbformat.read(f, as_version=4)

# Create a new Word document
doc = Document()
doc.add_heading("Extracted Code from Notebook", level=1)

# Extract and add code cells to the document
for cell in nb['cells']:
    if cell['cell_type'] == 'code':
        doc.add_heading("Code Cell", level=2)
        doc.add_paragraph(cell['source'])

# Save the document
doc.save("C:/Users/CHUMKI/Desktop/AA/multiplemodcode.docx")

print(f"Notebook code saved to:
{"C:/Users/CHUMKI/Desktop/AA/multiplemodcode.docx"}")
```

Code Cell