Configuration Manual

MSc Research Project
Data Analytics

Ram Abhilash Vasamsetti
Student ID: x22117491

School of Computing
National College of Ireland

Supervisor: Athanasios Staikopoulos

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ram Abhilash Vasamsetti
Student ID:	x22117491
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Athanasios Staikopoulos
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	682
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ram Abhilash Vasamsetti
x22117491

1 Introduction

This Configuration Manual describes the hardware and software requirements and necessary configurations utilized in the research project "Using Time Series Predictive Models for Early Detection of Gambling Addiction in Problem Gamblers"

The manual is divided into 4 main sections. Section 2 gives the overview of the Research project. The section 3 highlights the Hardware and Software Prerequisites. The next Section 4 elaborates the implementation requirements where necessary libraries are discussed. The section 5 gives as overview of the dataset and its import. Section 6 briefs about the models and any necessary configurations.

2 Research Overview

This research project focuses on using Time series predictive models such as ARIMA, SARIMA and LSTM to forecast the future betting patterns of Problem Gamblers, Thereby detecting Gambling Addiction. The work uses K means clustering for detection of problem gamblers and forecasting models to predict their future betting patterns.

3 System Prerequisites

3.1 Hardware Prerequisites

The implementation is carried on the windows system with the following configuration :

Processor	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz 2.50 GHz
Installed RAM	16.0 GB (15.9 GB usable)
System type	64-bit operating system, x64-based processor

Table 1: Operating System configuration

3.2 Software Prerequisites

Jupyter Notebook from Anaconda with Python Version: 3.11.3 has been for the implementation for this Research project.

4 Implementation Requirements

The Fig. 1 presents all the python libraries used in the research project. Any libraries missing in the machine are installed via the command "pip install [library name]".

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from pandas.plotting import parallel_coordinates
from ipywidgets import interact, DatePicker
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from statsmodels.tsa.arima.model import ARIMA
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.optimizers import Adam

```

Figure 1: Required Libraries for implementation

5 Dataset

5.1 Dataset Source

The dataset have been obtained from the Transparency Project. A Harvard Medical university initiative for encouraging research of Addictions (Division on Addiction; 2021). The dataset used has been contributed to The Transparency Project via (Gray et al.; 2012) research on Responsible Gambling. The folder contains 5 files as in Table 2

Raw Datset I.Demographics_Gray_LaPlante_PAB_2012.dat
Raw Datset II.Daily aggregates_Gray_LaPlante_PAB_2012.dat
Raw Datset III.Responsible gambling details_Gray_LaPlante_PAB_2012.dat
CodeBook_for Gray_LaPlante_PAB_2012 (Variable Definitions)
AnalyticDataset_Gray_LaPlante_PAB_2012.dat (Not Used for this Research)

Table 2: Data Source Contents

Out of all the dataset source files, Only Raw Dataset I (Demographic Information), Raw Dataset II (Daily Aggregates) and Raw Dataset III (Responsible Gambling details) are imported into the dataframes. The CodeBook file contains the legend and column definitions for each dataset used. The Analytics Dataset is a comprehensive dataset collected by the primary researcher. As per the research objectives, dedicated analytics have been performed in the data gathering and transformation phase and, thus, this file has not been used.

5.2 Dataset Import

The code as indicated in Fig. 2 is used with delimiter as `\t` . All the three raw datasets are imported. The path of the datasets need to be specified as mentioned in Fig. 2 where datasets are stored in a folder called [dataset]. Please ensure the datasets are not tampered by manually copy pasting the data.

```
daily_agg_df = pd.read_csv('./datasets/Raw Datset II.Daily aggregates_Gray_LaPlante_PAB_2012.dat', delimiter='\t')
rg_det_df = pd.read_csv('./datasets/Raw Datset III.Responsible gambling details_Gray_LaPlante_PAB_2012.dat', delimiter='\t')
demog_df = pd.read_csv('./datasets/Raw Datset I.Demographics_Gray_LaPlante_PAB_2012.dat', delimiter='\t')


```

Figure 2: Code for importing datasets to the project

5.3 Merging of Dataset

The datasets upon minor transformations on datetime, are merged to simplify the data relation. The Fig. 3 shows the final merged dataframe contaning all three dataframes. Tthe product type =='2' indicates casino games from the codebook. This parameter can be changed to identify forecasts in other games played by different user.

Merging of Datasets

In [13]:
```
merged_df = daily_agg_df.merge(demog_df, on='UserID', how='outer')
merged_df = merged_df.merge(rg_det_df, on='UserID', how='outer')

```

In [14]:
```
merged_df['RG_case'].value_counts()
```

Out[14]:
```
1     811570
0     170233
Name: RG_case, dtype: int64
```

In [15]:
```
merged_df
```

Out[15]:

	UserID	ProductType	Turnover	Hold	NumberofBets	Aggregate_Date	RG_case	CountryName	LanguageName	Gender	YearofBirth	Registration_d
0	31965	1.0	15.3388	15.3388	1	2000-05-08	1	19	8	1	1971	1999-09
1	31965	1.0	34.1594	34.1594	5	2000-05-10	1	19	8	1	1971	1999-09
2	31965	1.0	24.5419	24.5419	4	2000-05-18	1	19	8	1	1971	1999-09
3	31965	1.0	2.5309	2.5309	1	2000-05-22	1	19	8	1	1971	1999-09
4	31965	1.0	15.3387	15.3387	2	2000-05-23	1	19	8	1	1971	1999-09
...	...	...	...	...	...	...	...	...	...	...	...	

Figure 3: Code for merging datasets

5.4 Feature Selection

The features are checked for correlation and highly correlated items are dropped by picking only the low correlated columns in the K - means clustering. The Fig. 4 shows the correlated columns. More columns can be dropped to find optimal features for the project (BUYRUKOĞLU and AKBAŞ; 2022). This paper has only excluded columns which are highly correlated.

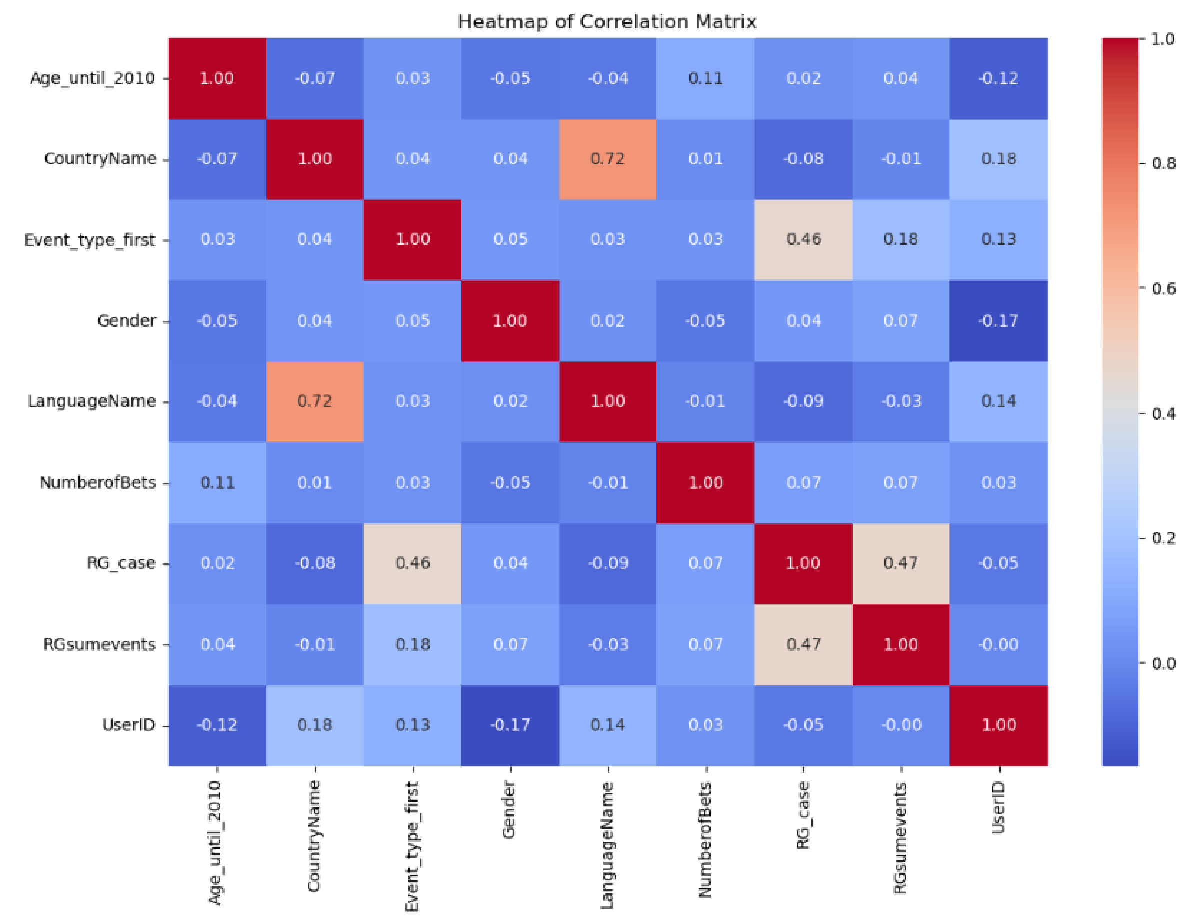

Figure 4: Heatmap for merged Dataset

6 Model Fitting

4 models have been used in total for the research project. Configurations for each model are given below.

6.1 K - Means Clustering

K - means clustering in the Fig. 5 require the data to be pivoted and features added as layers to 3d dataframe (Kobylin and Lyashenko; 2020). All the low correlated columns are passed through the [user data 3d] column. This values can be configured as per need.

K means Evaluation

```
In [24]:
inertias = []
for k_value in range(1, 6):
    model = KMeans(n_clusters=k_value, random_state=0)
    model.fit(user_data_array)
    inertias.append(model.inertia_)

plt.plot(range(1, 6), inertias, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
```

```
C:\Users\abhiv\AppData\Local\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\abhiv\AppData\Local\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\abhiv\AppData\Local\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\abhiv\AppData\Local\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\abhiv\AppData\Local\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(

Out[24]: Text(0.5, 1.0, 'Elbow Method for Optimal K')
```

Figure 5: K - Means Clustering Elbow method code

The optimal K value is set as 3 using Elbow method as shows in the Fig. 6. This can be configured as per requirement if better K value is found for different game type apart from ”2”.

```

# Pivot the DataFrame to create a 3D array with entries as rows, features as columns, and dates as depth
user_data_3d = filtered_df.pivot(index='UserID', columns='Aggregate_Date', values=['Turnover', 'Hold', 'NumberofBets','Age_u

# Fill missing values with zeros (if any)
user_data_3d = user_data_3d.fillna(0)

# Convert empty strings (' ') to float values of zero (0)
user_data_3d = user_data_3d.replace(' ', 0).astype(float)
user_data_3d = user_data_3d.astype(float)

# Convert the pivot table to a NumPy array
user_data_array = user_data_3d.to_numpy()

#k fold value
k = 3

# Perform K-means clustering
model = KMeans(n_clusters=k, random_state=0)
y_pred = model.fit_predict(user_data_array)

# Apply PCA to reduce dimensionality to 2D
pca = PCA(n_components=2)
user_data_pca = pca.fit_transform(user_data_array)

# Visualize the clustered entries using PCA components
plt.figure(figsize=(8, 6))
scatter = plt.scatter(user_data_pca[:, 0], user_data_pca[:, 1], c=y_pred, cmap='viridis')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')

```

Figure 6: K - Means Clustering Model

Cluster Labels can be configured based on Cluster Analysis done on the mean hold and turnover as in Fig. 7.

In [28]:
```python
# Create a scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=agguser_allfeature, x='Mean_Hold', y='Mean_NumberofBets', hue='Cluster', palette='viridis', alpha=0.7)
plt.xlabel('Mean_Hold')
plt.ylabel('Mean_NumberofBets')
plt.title('Cluster Visualization based on Mean Hold and Mean Number of Bets')
#plt.legend(title='Cluster', loc='upper right')
legend_labels = ['Moderate Problem Gamblers','Early Players','Problem Gamblers']# [f'Cluster {info["cluster_num"]}' for info
legend = plt.legend(handles=scatter.legend_elements()[0], title='Cluster', labels=legend_labels)

plt.show()
```

Figure 7: K - Means Clustering Label code

After Cluster Analysis. The moderate cluster user id need to be populated in the variable shown in Fig. 8.

In [34]:
```python
# features to consider : turnover, hold , number of bets for time series prediction
```

In [35]:
```python
moderate_pg_players= [868583, 1175809, 1411743, 1457496, 1486136, 1662632, 1679490, 1776178, 1790848, 1921204, 2070894, 2150

count = len(moderate_pg_players)
print("Count of moderate addicted players:", count)

```

```
Count of moderate addicted players: 40
```

Figure 8: K - Means Moderate players config.

6.2 ARIMA/ SARIMA

Before Fitting into ARIMA/ SARIMA models the moderate cluster id needs to be segregated into stationary and non stationary (Franses; 1991). For this ADF test is being used. The threshold can be configured as in Fig. 9. A confidence interval of 0.05 is being used as default to reject the null hypothesis.

the [is_stationary_df] is a df which stores the results of ADF test and is used to segregate into ARIMA/ SARIMA or LSTM model data input.

Both ARIMA and SARIMA models in Fig. 10 and 11 are implemented over a loop for each user. Thus both models are coded in the same loop for efficiency. SARIMA model is implemented using SARIMAX. It is SARIMA model with exogenous factors. The ARIMA/SARIMAX parameters can be configures based on seasonality and repetitive trends identified in the manual examination of the usage plot (Kumar Dubey et al.; 2021).

6.3 LSTM

LSTM model is used at the last and is implemented for non stationray datapoints (Kumar Dubey et al.; 2021). The Hyperparameter tuning section in the Fig. 12 is commented out intentionally to save time. It can be un-commented and be used only if there is change in the dataset. The LSTM model has already undergone hyper parameter tuning and the results are commented for reference as can be seen in the Fig. 13. The best parameters obtained are directly used in the LSTM implementation over a loop to forecast for each user as seen in Fig. 14.

Filtering of data based on hypothesis testing for Stationarity

In [37]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller

# Create a dictionary to store user IDs, their corresponding 'is_stationary' values, and 'hypothesis_test_result'
is_stationary_data = {'UserID': moderate_pg_players, 'is_stationary': [], 'hypothesis_test_result': []}

# Iterate through the list of user IDs
for user_id in moderate_pg_players:
    # Extract the user's turnover data
    user_turnover = user_data_3d['Turnover'][user_data_3d.index == user_id].values.ravel()

    # Perform the ADF test for stationarity
    result = adfuller(user_turnover)

    # Check if the time series is stationary based on the p-value
    if result[1] <= 0.05:
        is_stationary = 1  # Stationary
        hypothesis_test_result = 'Reject Null Hypothesis'  # Stationarity is significant
    else:
        is_stationary = 0  # Not stationary
        hypothesis_test_result = 'Fail to Reject Null Hypothesis'  # Stationarity is not significant

    # Append the 'is_stationary' and 'hypothesis_test_result' values to the list
    is_stationary_data['is_stationary'].append(is_stationary)
    is_stationary_data['hypothesis_test_result'].append(hypothesis_test_result)

# Create a new DataFrame from the dictionary
is_stationary_df = pd.DataFrame(is_stationary_data)

# Print the new DataFrame
print(is_stationary_df)
```

Figure 9: Stationarity Test

```python
    # Scale the data
    scaler = MinMaxScaler(feature_range=(0, 1))
    x_single_user_turnover_ravel_scaled = scaler.fit_transform(x_single_user_turnover_ravel.reshape(-1, 1))

    # Split the data into training and test sets
    split_ratio = 0.97  # 97% for training, 3% for testing
    split_index = int(len(x_single_user_turnover_ravel) * split_ratio)

    train_data = x_single_user_turnover_ravel_scaled[:split_index]
    test_data = x_single_user_turnover_ravel_scaled[split_index:]

    # Create a time index for your data
    time_index_train = pd.date_range(start=start_date, periods=len(train_data), freq=frequency)

    # Create a time index for your data
    time_index_test = pd.date_range(start=time_index_train[-1], periods=len(test_data), freq=frequency)

    #ARIMA

    # Define the ARIMA model
    model_arima = ARIMA(train_data, order=(1, 1, 1))

    # Fit the ARIMA model to the training data
    FITmodel_arima = model_arima.fit()

    # Forecast the test series using ARIMA
    FITmodel_arima_forecast = FITmodel_arima.predict(start=split_index, end=len(x_single_user_turnover_ravel) - 1)

    # Inverse scale the ARIMA forecasted values
    FITmodel_arima_forecast = scaler.inverse_transform(FITmodel_arima_forecast.reshape(-1, 1)).reshape(-1)
```

Figure 10: ARIMA Model

```
#SARIMA

# Define the SARIMA model with seasonal difference and order
model_sarima_monthly = SARIMAX(train_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 14))

# Fit the model to the training data
FITmodel_sarima_monthly = model_sarima_monthly.fit()

# Forecast the test series
FITmodel_sarima_monthly_forecast = FITmodel_sarima_monthly.forecast(steps=len(test_data))

# Inverse scale the forecasted values
FITmodel_sarima_monthly_forecast = scaler.inverse_transform(FITmodel_sarima_monthly_forecast.reshape(-1, 1)).reshape(-1)

# Inverse scale the training data
train_data_inverse = scaler.inverse_transform(train_data.reshape(-1, 1)).reshape(-1)

# Inverse scale the test data
test_data_inverse = scaler.inverse_transform(test_data.reshape(-1, 1)).reshape(-1)
```

Figure 11: SARIMA Model

```
# Define hyperparameters for tuning
units_values = [64, 128, 256]
learning_rate_values = [0.01, 0.001, 0.0001]

best_rmse = float('inf')
best_params = None

# Perform grid search
for units in units_values:
    for learning_rate in learning_rate_values:
        model = create_lstm_model(units, learning_rate)

        # Train the model
        model.fit(x_train, y_train, batch_size=1, epochs=3, verbose=0)

        # Prepare the testing data
        test_data = scaled_data[training_data_monthly_len - 7 * test_weeks:, :]
        x_test = []

        for i in range(7 * test_weeks, len(test_data)):
            x_test.append(test_data[i - (7 * test_weeks):i, 0])

        x_test = np.array(x_test)
        x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

        # Get predictions
        predictions = model.predict(x_test)
        predictions = scaler.inverse_transform(predictions)
```

Figure 12: Hyperparameter Tuning

```
In [52]:
# Mean: 35.68018188057653
# Standard Deviation: 122.09556811748713
# Variance: 14907.32775393194
# 1/1 [==============================] - 1s 853ms/step
# Units: 64, Learning Rate: 0.01, Test RMSE: 375.2553253207926
# 1/1 [==============================] - 1s 840ms/step
# Units: 64, Learning Rate: 0.001, Test RMSE: 231.66883181925292
# 1/1 [==============================] - 1s 907ms/step
# Units: 64, Learning Rate: 0.0001, Test RMSE: 299.90232400037746
# 1/1 [==============================] - 1s 902ms/step
# Units: 128, Learning Rate: 0.01, Test RMSE: 398.97323688410444
# 1/1 [==============================] - 1s 879ms/step
# Units: 128, Learning Rate: 0.001, Test RMSE: 241.82620950011136
# 1/1 [==============================] - 1s 854ms/step
# Units: 128, Learning Rate: 0.0001, Test RMSE: 279.9525872191042
# 1/1 [==============================] - 1s 886ms/step
# Units: 256, Learning Rate: 0.01, Test RMSE: 398.22153453188594
# 1/1 [==============================] - 1s 876ms/step
# Units: 256, Learning Rate: 0.001, Test RMSE: 289.69737162552474
# 1/1 [==============================] - 1s 879ms/step
# Units: 256, Learning Rate: 0.0001, Test RMSE: 266.59826253364014
# Best Hyperparameters: {'units': 64, 'learning_rate': 0.001}, Best RMSE: 231.66883181925292
```

Figure 13: Results of Hyperparameter Tuning saved in comments to reduce computation time

```python
# Define the best hyperparameters
best_units = 64
best_learning_rate = 0.001

# Build and compile the LSTM model with the best hyperparameters
best_model = Sequential()
best_model.add(LSTM(units=best_units, return_sequences=True, input_shape=(x_train.shape[1], 1)))
best_model.add(LSTM(units=best_units, return_sequences=False))
best_model.add(Dense(units=25))
best_model.add(Dense(units=1))

optimizer = Adam(learning_rate=best_learning_rate)
best_model.compile(optimizer=optimizer, loss='mean_squared_error')

# Train the model
best_model.fit(x_train, y_train, batch_size=1, epochs=1)  # Adjust epochs as needed

# Prepare the testing data
test_data = scaled_data[training_data_monthly_len - (7*test_weeks):, :]
x_test = []

for i in range(7*test_weeks, len(test_data)):
    x_test.append(test_data[i-(7*test_weeks):i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

# Get predictions
predictions = best_model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

Figure 14: LSTM model

References

BUYRUKOĞLU, S. and AKBAŞ, A. (2022). Machine learning based early prediction of type 2 diabetes: A new hybrid feature selection approach using correlation matrix with heatmap and sfs, *Balkan Journal of Electrical and Computer Engineering* **10**: 110–117.

Division on Addiction, Cambridge Health Alliance, a. H. M. S. t. h. (2021). Behavioral characteristics of internet gamblers who trigger corporate responsible gambling interventions, Medford, MA: Division on Addiction, The Transparency Project [database distributor].

Franses, P. H. (1991). Seasonality, non-stationarity and the forecasting of monthly time series, *International Journal of Forecasting* **7**(2): 199–208.
URL: *https://www.sciencedirect.com/science/article/pii/016920709190054Y*

Gray, H. M., LaPlante, D. A. and Shaffer, H. J. (2012). Behavioral characteristics of Internet gamblers who trigger corporate responsible gambling interventions., *Psychology of Addictive Behaviors* **26**(3): 527–535.

Kobylin, O. and Lyashenko, V. (2020). Time series clustering based on the k-means algorithm, *Journal La Multiapp* **1**: 1–7.

Kumar Dubey, A., Kumar, A., García-Díaz, V., Kumar Sharma, A. and Kanhaiya, K. (2021). Study and analysis of sarima and lstm in forecasting time series data, *Sustainable Energy Technologies and Assessments* **47**: 101474.
URL: *https://www.sciencedirect.com/science/article/pii/S2213138821004847*