Ecole Polytechnique de Thiès
Département Génie Informatique et Télécommunication

Modélisation Stochastique

A New Delay History Predictor for Multi-skill Call Center

Application to the VANAD Call Center

DIC2-GIT, 2022-2023, M. Michel Seck

PLAN

Importation des bibliothèques nécessaires ⏮👆⏭

In [1]:
```
from datetime import datetime
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import glob
from vanad.preprocess import load_dataset
from sklearn.model_selection import train_test_split
```

Importation du dataset ⏮👆⏭

In [2]:
```
# importation du dataset
dataset = pd.read_csv("data.csv")
#dataset = load_dataset()
```

In [37]:
```
# get info about the dataset
dataset.info()
```

In [38]:
```
# get info about the name of columns
dataset.columns
```

In [5]:
```
# show some lines
dataset.head()
```

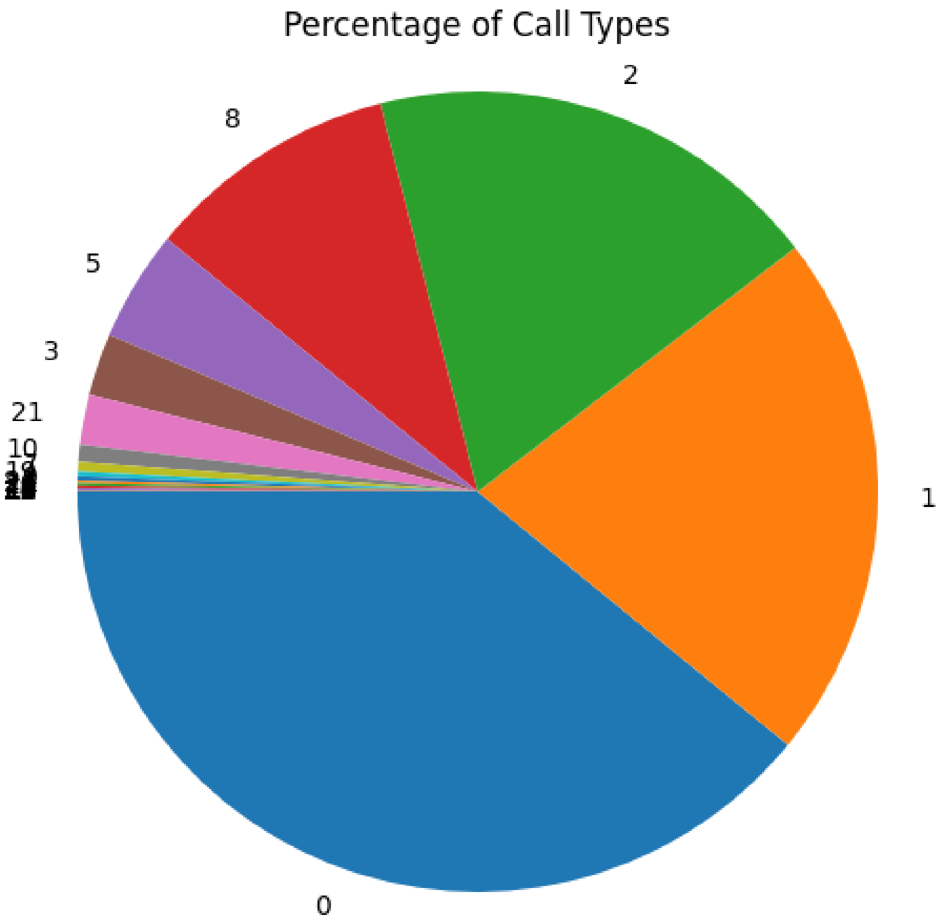
Out[5]:

	Type	Is_Served	Arrival_Time	Service_Time	Number_Of_Server	Wait_List_Length
0	2	1	3633	20.0	1	
1	2	1	3771	82.0	2	
2	0	1	3853	155.0	4	
3	1	1	3892	116.0	6	
4	1	1	3838	275.0	4	

5 rows × 37 columns

Identify & Select most descriptive features ⏮👆🏽⏭

In [6]:
```
# Calculate the percentage of each call type
call_type_percentages = dataset['Type'].value_counts(normalize=True) * 100

# Create a pie chart
plt.figure(figsize=(8, 6))
plt.pie(call_type_percentages, labels=call_type_percentages.index, startangl
plt.title('Percentage of Call Types')
plt.axis('equal')  # Equal aspect ratio ensures that the pie chart is circul
plt.show()
```



In [7]:
```
np.sum(call_type_percentages[:5])
```

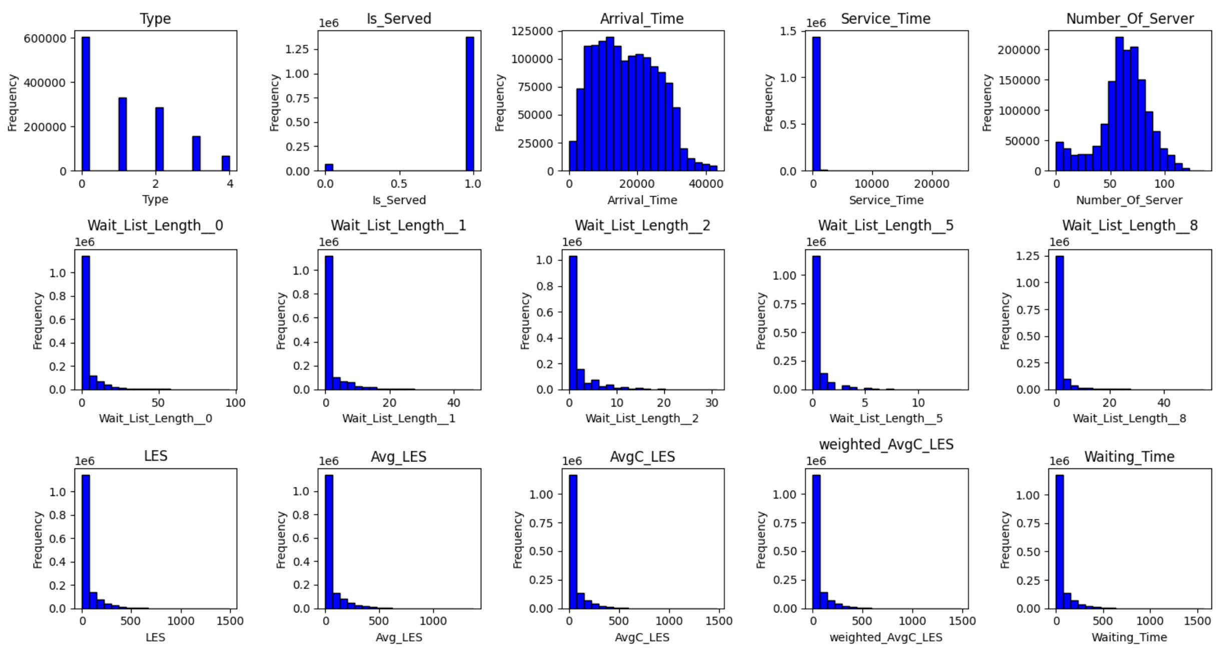
Out[7]:
```
93.58906876554941
```

In [8]:
```
# Filter by types 1, 2, 3, and 4
filtered_dataset = dataset[dataset['Type'].isin([0, 1, 2, 8, 5])]

# Define the mapping dictionary
mapping = {0: 0, 1: 1, 2: 2, 8: 3, 5: 4}

# Remap values in the 'type' column
filtered_dataset.loc[:, 'Type'] = filtered_dataset['Type'].replace(mapping)

# Drop specific columns using del
columns_to_drop = [f'Wait_List_Length__{i}' for i in range(27) if i not in [
for column in columns_to_drop:
    del filtered_dataset[column]

filtered_df = filtered_dataset[
  (filtered_dataset['Is_Served'] != 0) &
  (filtered_dataset['Waiting_Time'] > 0)
]
```

In [39]:
```
filtered_dataset.info()
```

Feature Scaling ⏮️👆🏽⏭️

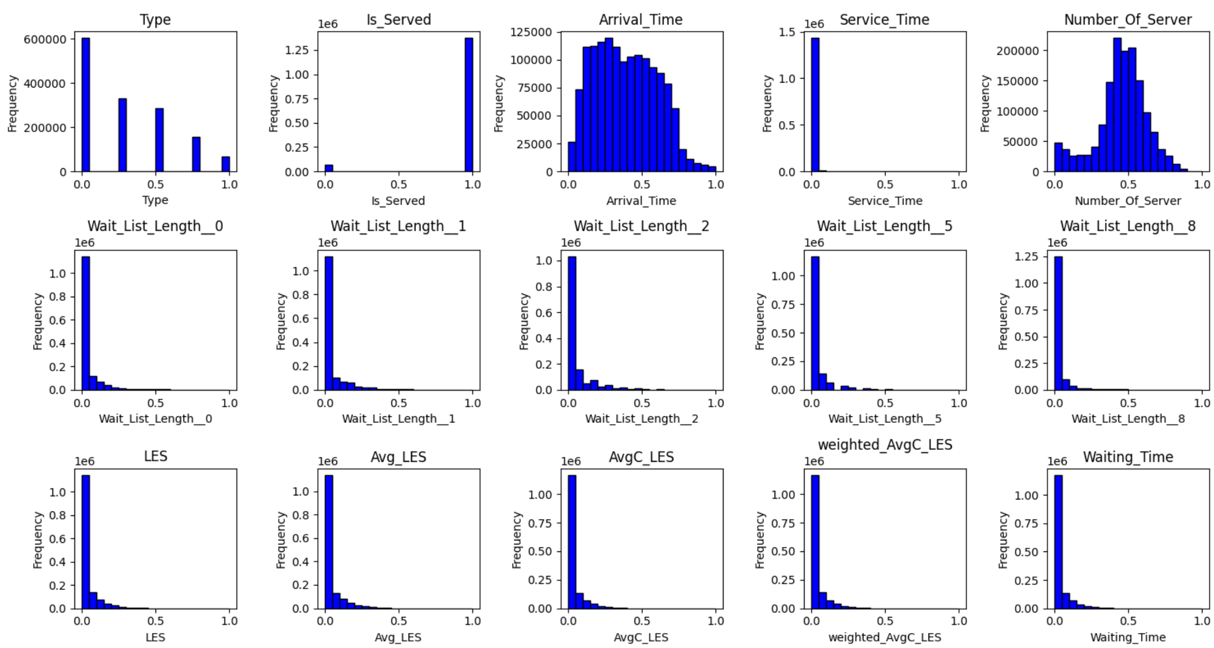
In [10]:
```
# Define the number of rows and columns for the subplots
num_rows = 3
num_cols = 5

# Get the feature column names (excluding the target column)
feature_columns = filtered_dataset.columns

# Calculate the number of subplots
num_subplots = len(feature_columns)

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 8))
# Iterate over feature columns and create histograms on subplots
for i, column in enumerate(feature_columns):
    if i >= num_rows * num_cols:
        break  # Stop creating subplots after filling the grid
    row_idx = i // num_cols
    col_idx = i % num_cols
    axes[row_idx, col_idx].hist(filtered_dataset[column], bins=20, color='bl
    axes[row_idx, col_idx].set_title(column)
    axes[row_idx, col_idx].set_xlabel(column)
    axes[row_idx, col_idx].set_ylabel('Frequency')

# Adjust layout for subplots
plt.tight_layout()
plt.show()
```



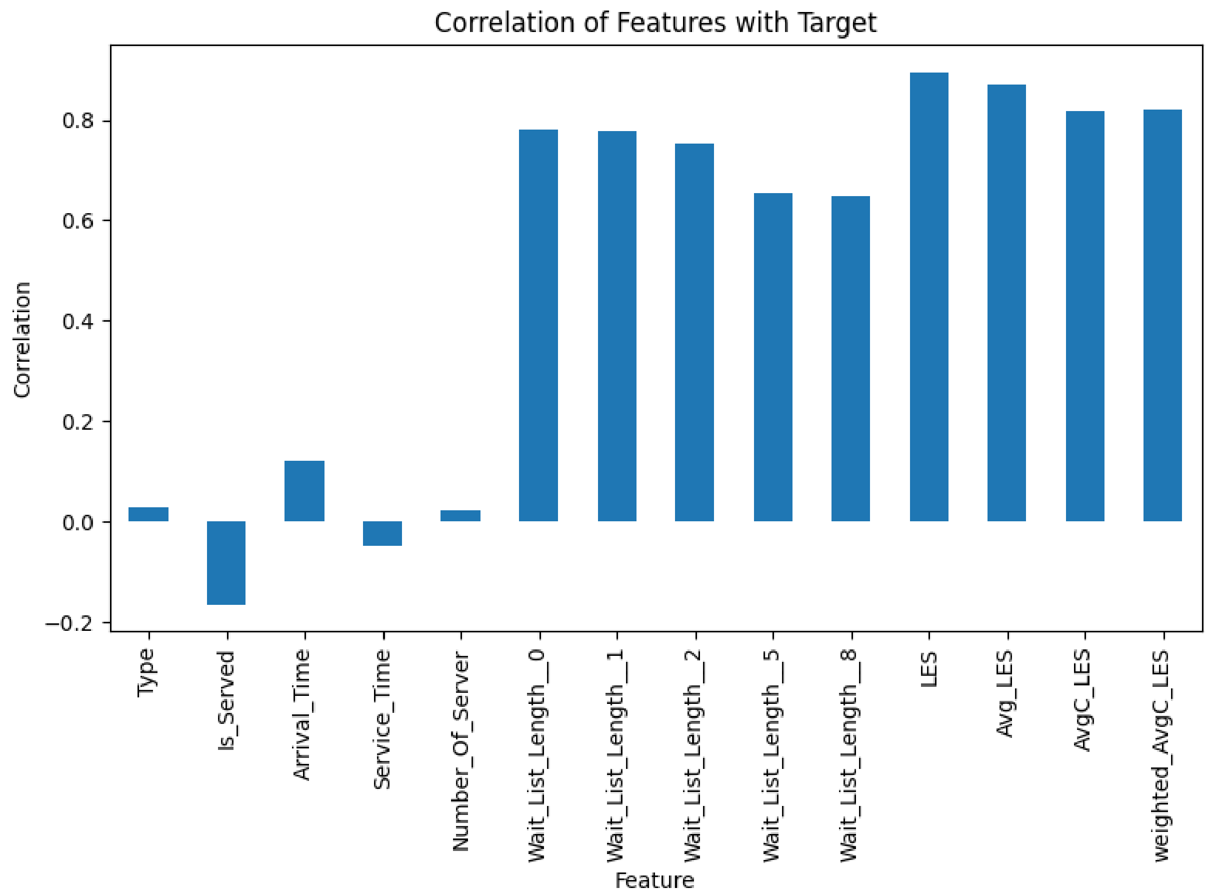
In [11]:
```
normalized_dataset = (filtered_dataset - filtered_dataset.min()) / (filterec
```

In [12]:
```
# Define the number of rows and columns for the subplots
num_rows = 3
num_cols = 5
```

```
# Get the feature column names (excluding the target column)
feature_columns = normalized_dataset.columns

# Calculate the number of subplots
num_subplots = len(feature_columns)

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 8))
# Iterate over feature columns and create histograms on subplots
for i, column in enumerate(feature_columns):
    if i >= num_rows * num_cols:
        break  # Stop creating subplots after filling the grid
    row_idx = i // num_cols
    col_idx = i % num_cols
    axes[row_idx, col_idx].hist(normalized_dataset[column], bins=20, color='
    axes[row_idx, col_idx].set_title(column)
    axes[row_idx, col_idx].set_xlabel(column)
    axes[row_idx, col_idx].set_ylabel('Frequency')

# Adjust layout for subplots
plt.tight_layout()
plt.show()
```



In [13]:
```
# Calculate correlations
correlations = normalized_dataset.corr()['Waiting_Time'].drop('Waiting_Time'

# Plot correlations
plt.figure(figsize=(8, 6))
correlations.plot(kind='bar')
plt.title('Correlation of Features with Target')
plt.xlabel('Feature')
plt.ylabel('Correlation')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



In [14]:
```
normalized_dataset.head()
```

Out[14]:

	Type	Is_Served	Arrival_Time	Service_Time	Number_Of_Server	Wait_List_Length_
0	0.50	1.0	0.084084	0.000847	0.007353	
1	0.50	1.0	0.087279	0.003349	0.014706	
2	0.00	1.0	0.089177	0.006294	0.029412	
3	0.25	1.0	0.090080	0.004721	0.044118	
4	0.25	1.0	0.088830	0.011136	0.029412	

In [40]:
```
normalized_dataset.info()
```

Data splitting ⏮️👆🏽⏭️

In [16]:
```
# Split the dataset into train and test sets
X = normalized_dataset.drop(columns=['Waiting_Time'])  # Features
y = normalized_dataset['Waiting_Time']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

```python
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (1010634, 14)
X_test shape: (433129, 14)
y_train shape: (1010634,)
y_test shape: (433129,)
```

DL Model ⏮️👆🏽⏭️

In [17]:
```python
# Build the neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer for regression
])
```

In [26]:
```python
from tensorflow.keras import backend as K

# Define custom metric RRMSE
def rrmse(y_true, y_pred):
  mse = K.mean(K.square(y_true - y_pred))  # Mean Squared Error
  avg_wait_time = K.mean(K.square(y_true))  # Average wait time of N custome
  rrmse = K.sqrt(mse / avg_wait_time)  # Root Relative Mean Squared Error
  return rrmse * 100
```

In [28]:
```python
model.compile(
  optimizer=tf.keras.optimizers.legacy.Adam(),
  loss=tf.keras.losses.MeanSquaredError(),
  metrics=[
    tf.keras.metrics.RootMeanSquaredError(),
    rrmse
  ]
)
```

In [30]:
```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 64)                960       
                                                                 
 dense_1 (Dense)             (None, 32)                2080      
                                                                 
 dense_2 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 3,073
Trainable params: 3,073
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
history = model.fit(
  X_train,
  y_train,
  validation_split=0.2,
  epochs=15
).history
```

Performance Evaluation ⏮️👆🏽⏭️

In [33]:
```
y_pred = model.predict(X_test)
```

```
13536/13536 [==============================] - 3s 251us/step
```

In [34]:
```
# Combine X_test, y_true_df, and y_pred_df horizontally
combined_df = pd.concat(
  [
    X_test,
    pd.DataFrame(y_test.values.reshape(-1, 1), columns=['Waiting_Time'], inc
    pd.DataFrame(y_pred, columns=['Predicted_Time'], index=X_test.index)
  ],
  axis=1
)

combined_df.head()
```

Out[34]:

	Type	Is_Served	Arrival_Time	Service_Time	Number_Of_Server	Wait_List_
362515	0.25	1.0	0.262878	0.028244	0.580882	
1044525	0.75	1.0	0.124158	0.030181	0.176471	
465421	0.25	1.0	0.312536	0.013073	0.588235	
931096	0.50	1.0	0.687186	0.005326	0.330882	
318163	0.25	1.0	0.575831	0.014082	0.426471	

In [35]:
```
# Define custom metric RRMSE
def rrmse(y_true, y_pred):
  mse = np.mean(np.square(y_true - y_pred))  # Mean Squared Error
  avg_wait_time = np.mean(np.square(y_true)) # Average wait time of N custom
  rrmse = np.sqrt(mse / avg_wait_time)  # Root Relative Mean Squared Error
  return rrmse * 100
```

In [36]:
```
# Calculate RMSE and RRMSE for each group
grouped = combined_df.groupby('Type').apply(lambda group: {
    'LES': rrmse(group['Waiting_Time'], group['LES']),
    'Avg_LES': rrmse(group['Waiting_Time'], group['Avg_LES']),
    'AvgC_LES': rrmse(group['Waiting_Time'], group['AvgC_LES']),
    'W_AvgC_LES': rrmse(group['Waiting_Time'], group['weighted_AvgC_LES']),
    'ANN': rrmse(group['Waiting_Time'], group['Predicted_Time'])
}).reset_index()
```

```python
# Convert the dictionary-like values to separate columns
normalized_data = pd.json_normalize(grouped[0])

normalized_data
```

Out[36]:

	LES	Avg_LES	AvgC_LES	W_AvgC_LES	ANN
0	44.502269	51.609690	50.548024	50.122990	24.655293
1	45.593803	54.424435	53.330139	53.185670	26.624928
2	53.253891	61.611710	61.086576	60.666761	28.192417
3	49.715647	56.453244	54.785192	53.984475	25.918386
4	63.993984	68.378078	66.910578	66.953618	32.844168

Références ⏮️👆🏽⏭️

Here is a reference to the Python documentation.

Here are some references for more information on the libraries used:
🧑🏾‍💻 Pandas documentation
🧑🏾‍💻 NumPy documentation
🧑🏾‍💻 Matplotlib documentation
🧑🏾‍💻 Tensorflow documentation
🧑🏾‍💻 Sciki-learn documentation

Auteur ⏮️👆🏽⏭️

`Auteurs` : Mouhamadou Naby DIA & Abdou SAKHO

`Ecole` : Ecole Polytechnique de THIES

`Departement` : Genie Informatique et Telecoms

`Niveau` : DIC2