In [1]:
```
## Importing Basic Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import stats
from sklearn import metrics
from sklearn.model_selection import train_test_split
import random
import pickle
from sklearn.preprocessing import LabelEncoder
```

Visualizations

In [2]:
```
# Loading the Data
# Importing the data

df0=pd.read_csv('HARR.csv')
df0 = df0.dropna()
df0.head()
```

Out[2]:

	user	activity	timestamp	x-axis	y-axis	z-axis
0	1	Walking	4991922345000	0.69	10.80	-2.03
1	1	Walking	4991972333000	6.85	7.44	-0.50
2	1	Walking	4992022351000	0.93	5.63	-0.50
3	1	Walking	4992072339000	-2.11	5.01	-0.69
4	1	Walking	4992122358000	-4.59	4.29	-1.95

In [3]:
```
df0.info()
```

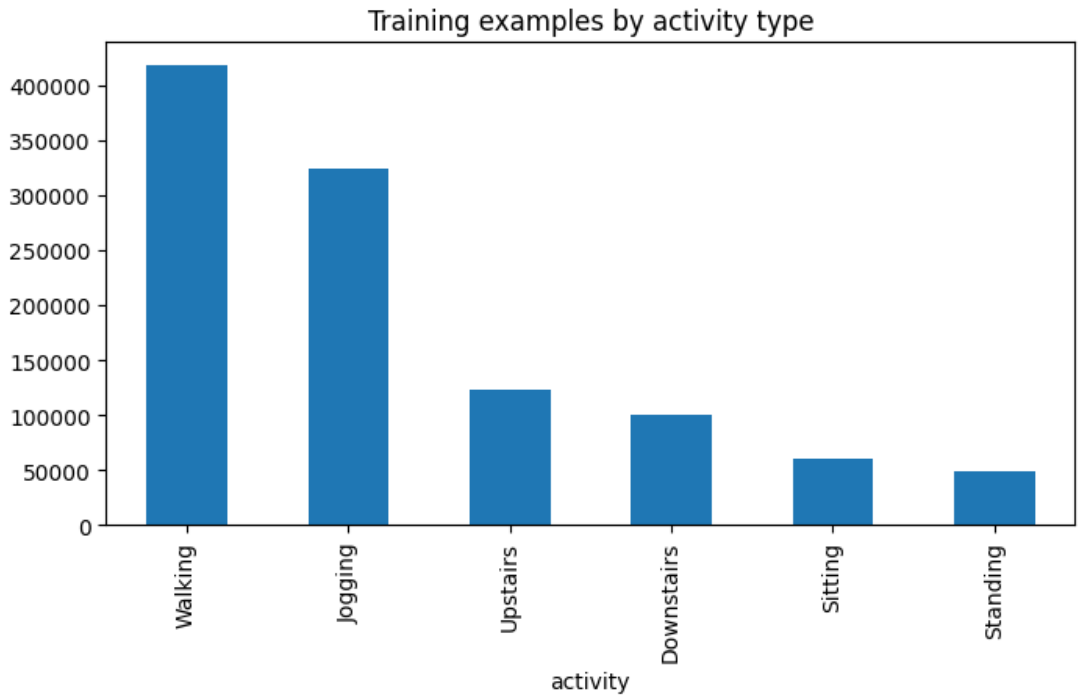
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1073623 entries, 0 to 1073622
Data columns (total 6 columns):
 #   Column     Non-Null Count    Dtype  
---  ------     --------------    -----  
 0   user       1073623 non-null  int64  
 1   activity   1073623 non-null  object 
 2   timestamp  1073623 non-null  int64  
 3   x-axis     1073623 non-null  float64
 4   y-axis     1073623 non-null  float64
 5   z-axis     1073623 non-null  float64
dtypes: float64(3), int64(2), object(1)
memory usage: 49.1+ MB
```

In [4]:
```
CountOfActivity = df0['activity'].value_counts()
print(CountOfActivity)
```

```
activity
Walking       417901
Jogging       324600
Upstairs      122598
Downstairs    100192
Sitting        59939
Standing       48393
Name: count, dtype: int64
```

In [5]:
```
CountOfActivity.plot(kind ='bar', title ='Training examples by activity
```

Out[5]:
```
<Axes: title={'center': 'Training examples by activity type'}, xlabel
='activity'>
```

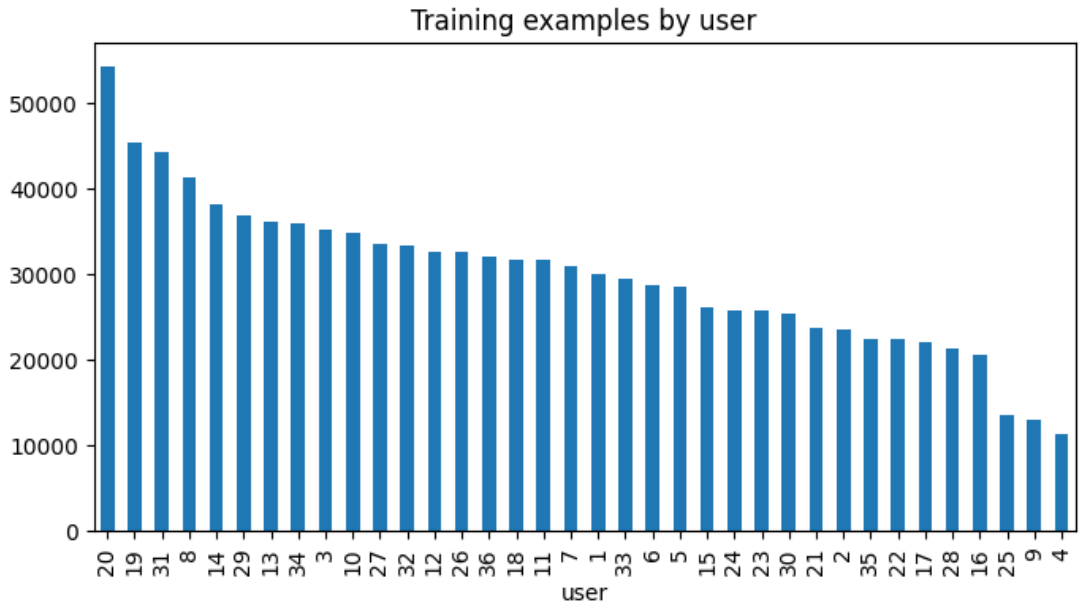


In [6]:
```
CountOfActivityPerPerson = df0['user'].value_counts()
print(CountOfActivityPerPerson)
totalActivity = CountOfActivityPerPerson.sum()
print("Total number of Activity recorded = " + str(totalActivity))
```

```
user
20    54294
19    45382
31    44282
8     41187
14    38192
29    36768
13    36063
34    35947
3     35158
10    34875
27    33456
32    33357
12    32641
26    32578
36    32108
18    31705
11    31658
7     30967
1     29978
33    29453
6     28703
5     28509
15    26082
24    25736
23    25673
30    25334
21    23703
2     23525
35    22394
22    22308
17    22020
28    21358
16    20469
25    13468
9     12923
4     11369
Name: count, dtype: int64
Total number of Activity recorded = 1073623
```

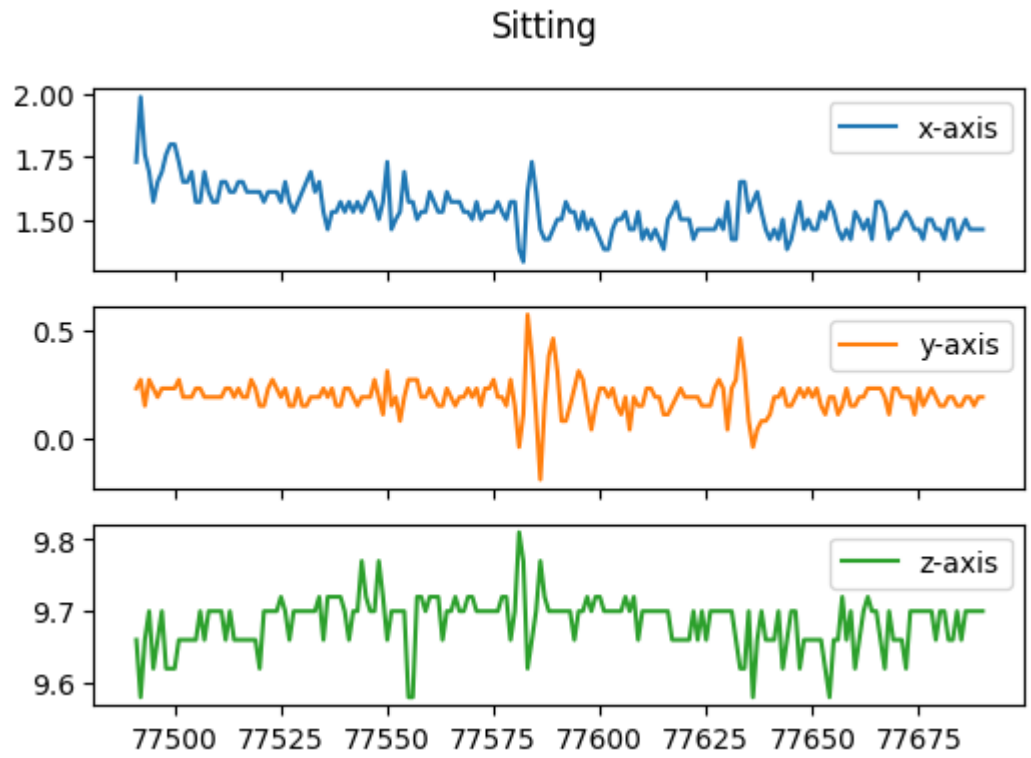
In [7]:
```
CountOfActivityPerPerson.plot(kind = 'bar', title = 'Training examples b
```

Out[7]:
```
<Axes: title={'center': 'Training examples by user'}, xlabel='user'>
```

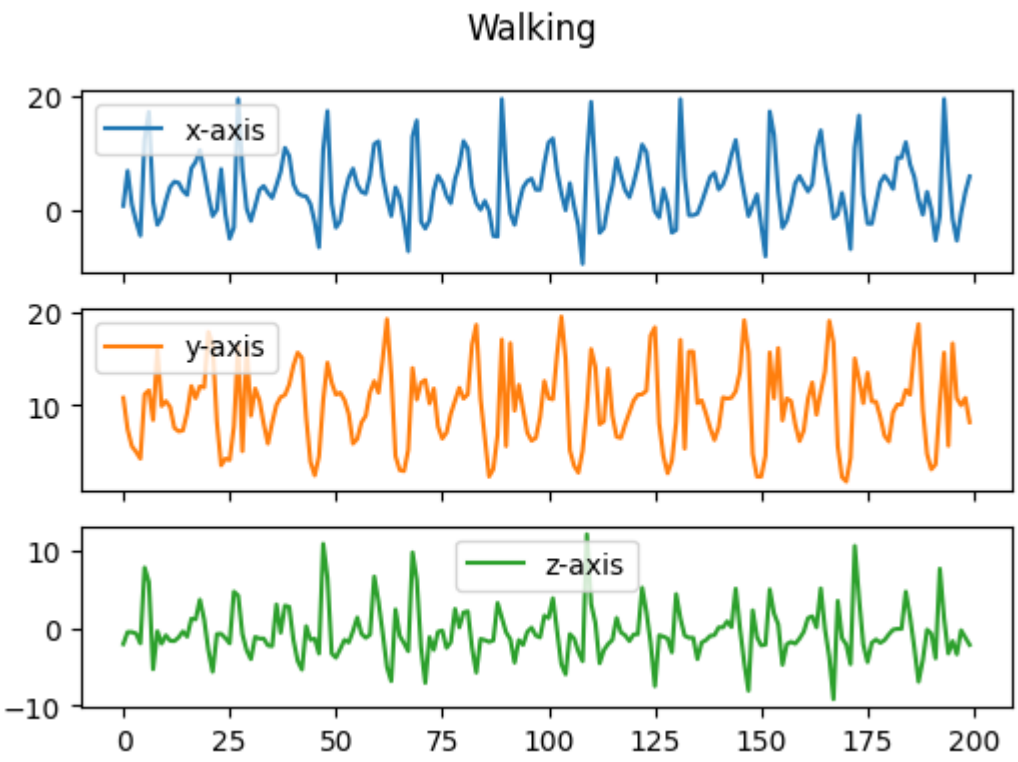


In [8]:
```
def plot_activity(activity,df0):
    ExtractRowsOfActivity = (df0['activity'] == activity)
    data = df0[ExtractRowsOfActivity]
    data = data[['x-axis','y-axis','z-axis']]
    data = data[:200]

    ax = data.plot(subplots=True, figsize=(6,4), title=activity)
```

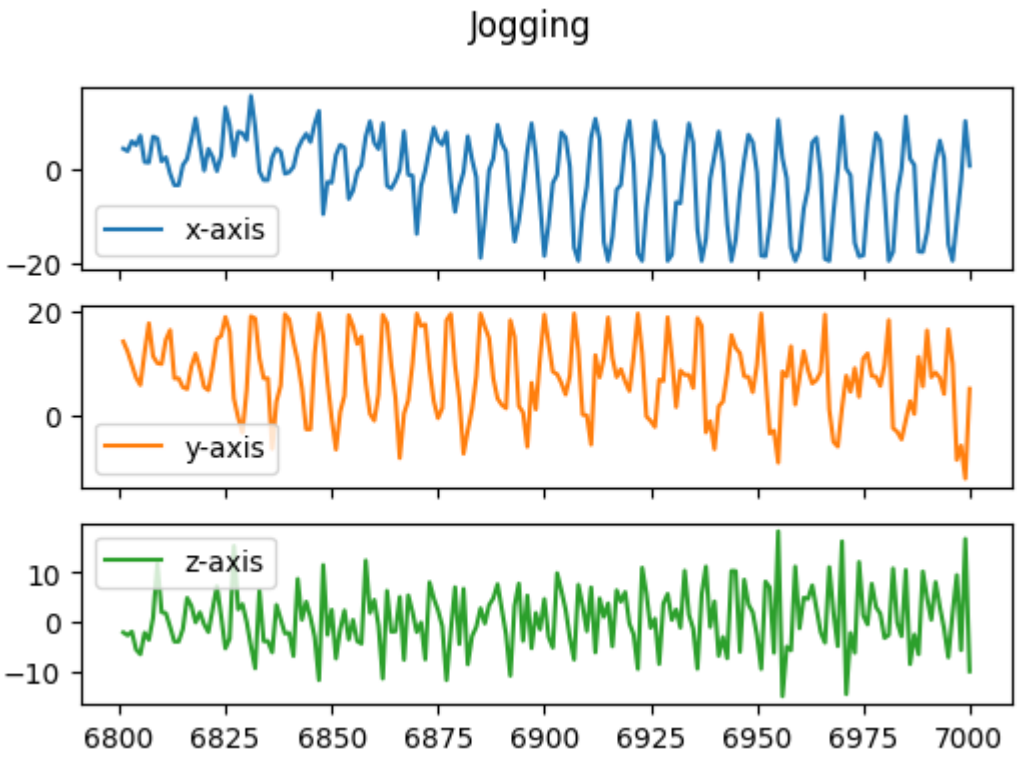
In [9]:
```
plot_activity('Sitting',df0)
```



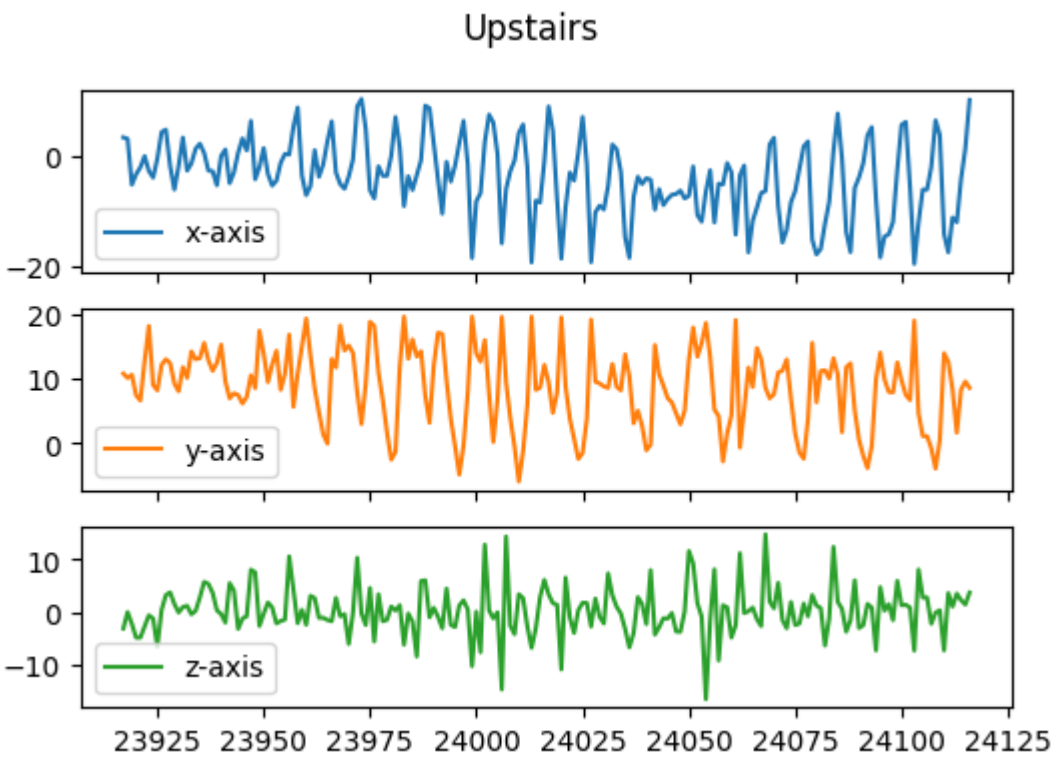
In [10]:
```
plot_activity('Walking',df0)
```



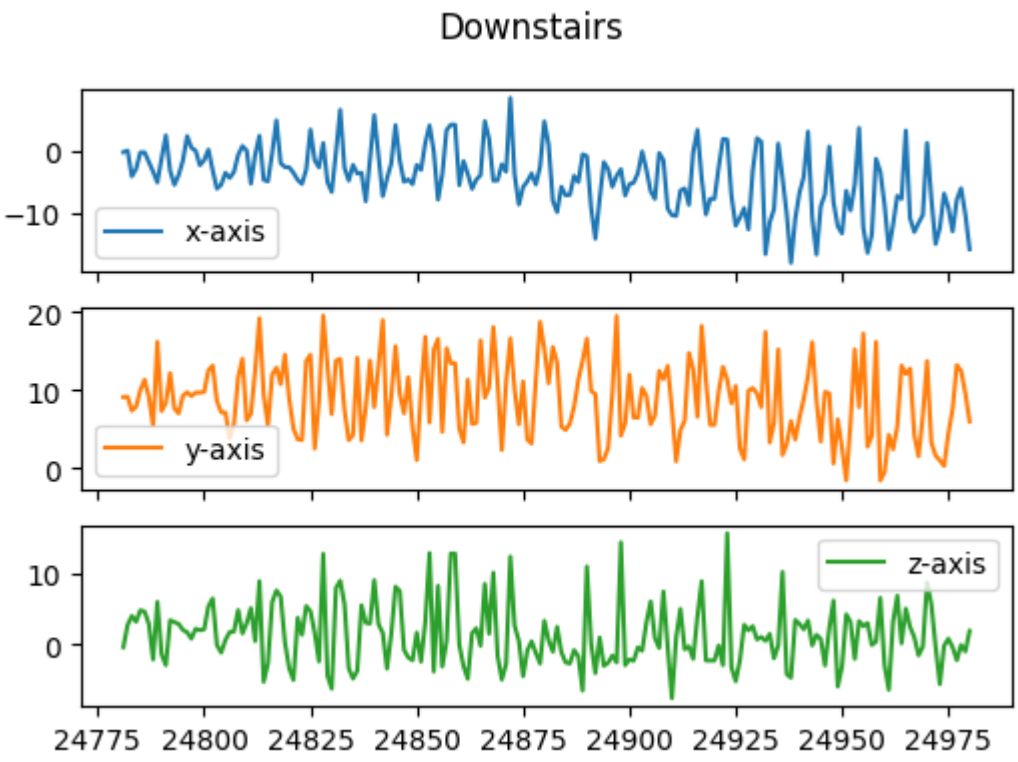
In [11]:
```
plot_activity('Jogging',df0)
```



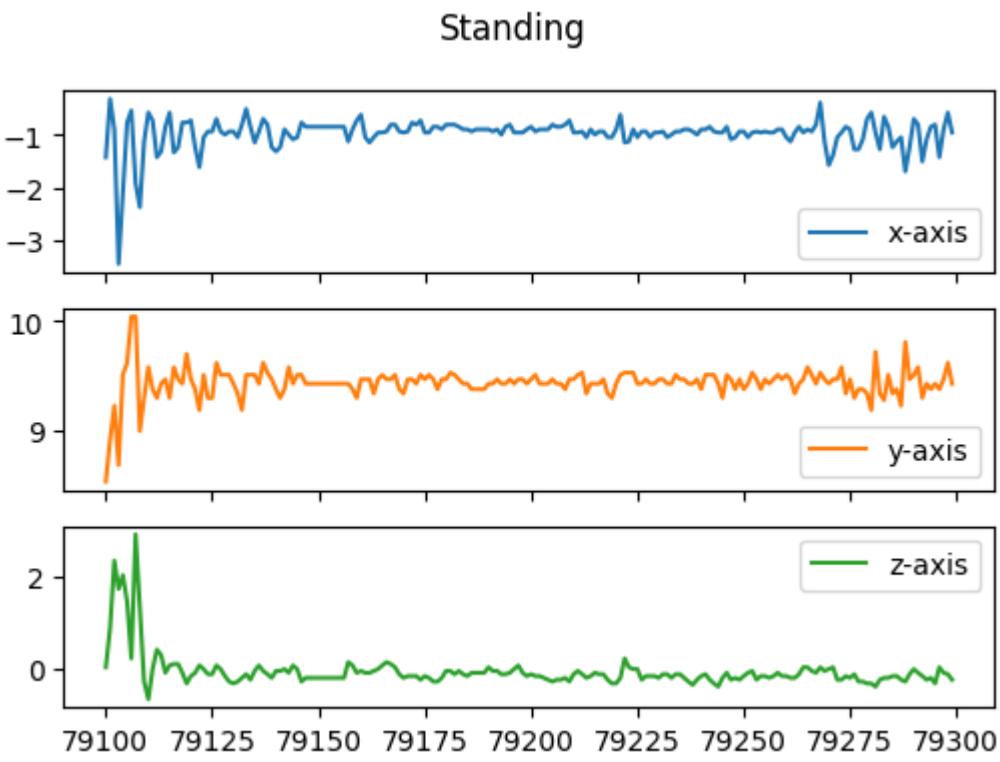
In [12]:
```
plot_activity('Upstairs',df0)
```



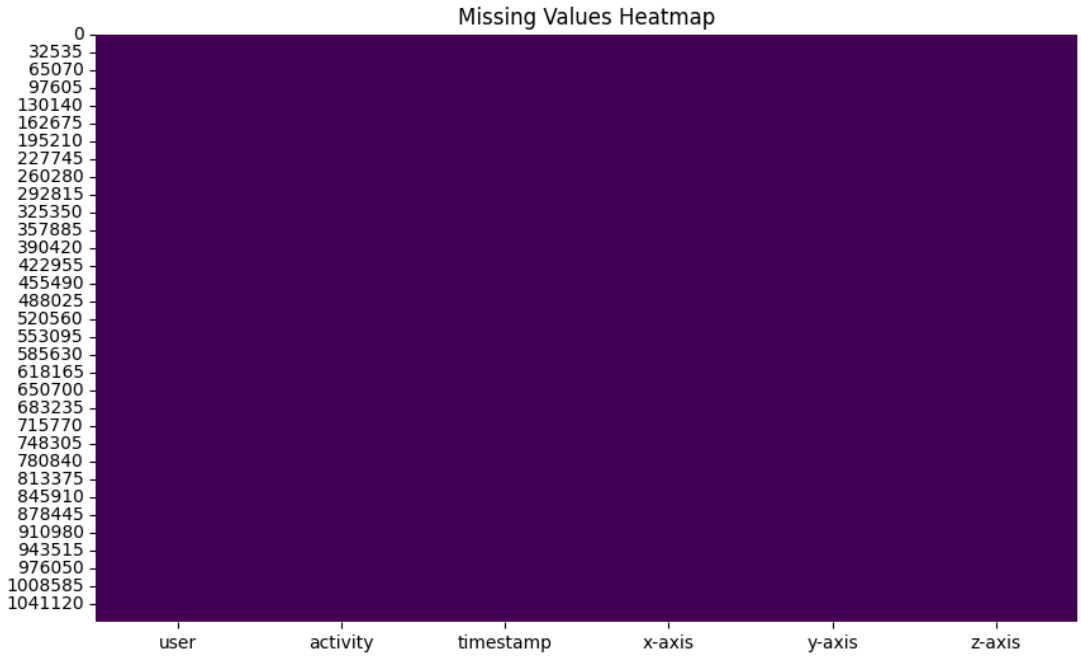
In [13]:
```
plot_activity('Downstairs',df0)
```



In [14]:
```
plot_activity('Standing',df0)
```



In [15]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(df0.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Values Heatmap')
plt.show()
```



In [ ]:
```

```

In [ ]:

```
```

In [16]:

```
df=pd.read_csv('HARR.csv')
df = df.dropna()
df.head()
```

Out[16]:

	user	activity	timestamp	x-axis	y-axis	z-axis
0	1	Walking	4991922345000	0.69	10.80	-2.03
1	1	Walking	4991972333000	6.85	7.44	-0.50
2	1	Walking	4992022351000	0.93	5.63	-0.50
3	1	Walking	4992072339000	-2.11	5.01	-0.69
4	1	Walking	4992122358000	-4.59	4.29	-1.95

EDA & FEATURE ENGINEERING

In [17]:

```
#Label Encoding
label_encoder = LabelEncoder()

df['activity_encoded'] = label_encoder.fit_transform(df['activity'])

print("Encoded Activities:")
print(df[['activity', 'activity_encoded']].drop_duplicates())

df = df.drop('activity', axis=1)
```

```
Encoded Activities:
         activity  activity_encoded
0         Walking                 5
6801      Jogging                 1
23917    Upstairs                 4
24781  Downstairs                 0
77491     Sitting                 2
79100    Standing                 3
```

In [18]:
```
# Assuming the timestamp is in Unix timestamp format
# Extract features from timestamps

df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ns')  # Change '

df['hour_of_day'] = df['timestamp'].dt.hour
df['day_of_week'] = df['timestamp'].dt.dayofweek

df = df.drop('timestamp', axis=1)

print("Modified DataFrame:")
print(df.head())
```

```
Modified DataFrame:
   user  x-axis  y-axis  z-axis  activity_encoded  hour_of_day  day_of_
week
0     1    0.69   10.80   -2.03                 5            1
3
1     1    6.85    7.44   -0.50                 5            1
3
2     1    0.93    5.63   -0.50                 5            1
3
3     1   -2.11    5.01   -0.69                 5            1
3
4     1   -4.59    4.29   -1.95                 5            1
3
```

In [19]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [20]:
```
X = df.drop('activity_encoded', axis=1)
y = df['activity_encoded']
```

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

Decision Tree

In [22]:
```
clf = DecisionTreeClassifier(random_state=42)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy_dt = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_dt)

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7557946210268949
Classification Report:
              precision    recall  f1-score   support

           0       0.42      0.42      0.42     20129
           1       0.82      0.83      0.83     65138
           2       0.99      0.99      0.99     12099
           3       0.96      0.96      0.96      9591
           4       0.50      0.50      0.50     24350
           5       0.80      0.80      0.80     83418

    accuracy                           0.76    214725
   macro avg       0.75      0.75      0.75    214725
weighted avg       0.76      0.76      0.76    214725
```

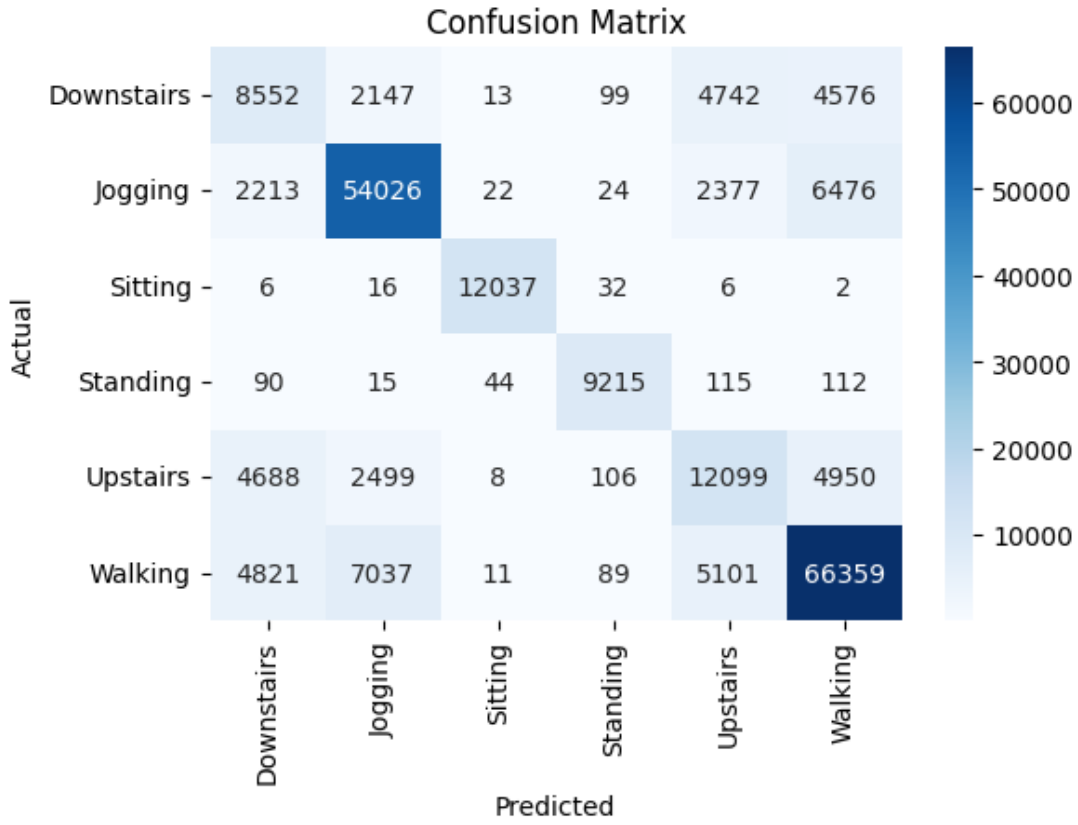
In [23]:
```
from sklearn.metrics import precision_score, recall_score, confusion_mat
import seaborn as sns
import matplotlib.pyplot as plt
```

In [24]:
```
#Precision and Recall

precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print("Precision:", precision)
print("Recall:", recall)
```

```
Precision: 0.7562957596942537
Recall: 0.7557946210268949
```

In [25]:

```
# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_enc
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [26]:
```
from sklearn.tree import export_text

# the inside of the Decision Tree

feature_names = ['activity','user','timestamp', 'x-axis', 'y-axis', 'z-a
tree_rules = export_text(clf, feature_names=feature_names, decimals=3)
print(tree_rules)
```

```
|--- timestamp <= 1.118
|   |--- x-axis <= 3.719
|   |   |--- user <= 7.950
|   |   |   |--- timestamp <= -1.098
|   |   |   |   |--- timestamp <= -4.331
|   |   |   |   |   |--- y-axis <= 0.500
|   |   |   |   |   |   |--- activity <= 12.000
|   |   |   |   |   |   |   |--- activity <= 8.500
|   |   |   |   |   |   |   |   |--- activity <= 4.500
|   |   |   |   |   |   |   |   |   |--- x-axis <= -6.280
|   |   |   |   |   |   |   |   |   |   |--- class: 5
|   |   |   |   |   |   |   |   |   |--- x-axis >  -6.280
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- activity >  4.500
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- activity >  8.500
|   |   |   |   |   |   |   |   |--- user <= -6.050
|   |   |   |   |   |   |   |   |   |--- class: 5
|   |   |   |   |   |   |   |--- user >  -6.050
```

In [ ]:

In [ ]:

XGB Classifier

In [27]:
```
df1=pd.read_csv('HARR.csv')
df1 = df1.dropna()
df1.head()
```

Out[27]:

	user	activity	timestamp	x-axis	y-axis	z-axis
0	1	Walking	4991922345000	0.69	10.80	-2.03
1	1	Walking	4991972333000	6.85	7.44	-0.50
2	1	Walking	4992022351000	0.93	5.63	-0.50
3	1	Walking	4992072339000	-2.11	5.01	-0.69
4	1	Walking	4992122358000	-4.59	4.29	-1.95

In [28]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df1['encoded_activity'] = label_encoder.fit_transform(df1['activity'])

print(df1['encoded_activity'].unique())
```

```
[5 1 4 0 2 3]
```

In [29]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

X1 = df1.drop(['activity', 'encoded_activity'], axis=1)  # Features
y1 = df1['encoded_activity']  # Target

X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.
```

In [30]:
```
from sklearn.preprocessing import MinMaxScaler

# Applying MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [31]:
```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score


xgb = XGBClassifier()

xgb.fit(X_train, y_train)

y_pred_xgb = xgb.predict(X_test)

accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print(f"Accuracy of XGBoost Classifier: {accuracy_xgb}")
```

```
Accuracy of XGBoost Classifier: 0.9498241937361742
```

In [32]:
```
print(f"Accuracy: {accuracy_xgb * 100:.2f}%")
```

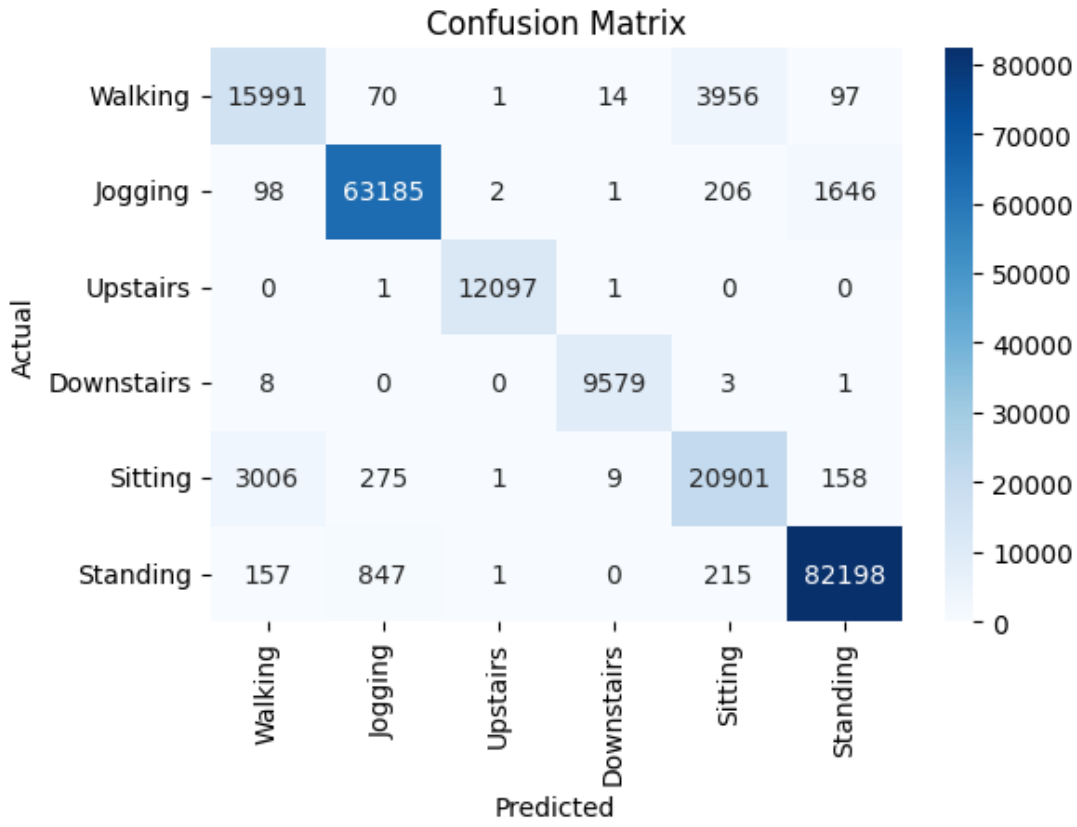
```
Accuracy: 94.98%
```

In [33]:
```
y_pred_xgb = xgb.predict(X_test)
```

In [34]:
```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

In [35]:
```
print("Classification Report:\n", classification_report(y_test, y_pred_x

conf_matrix = confusion_matrix(y_test, y_pred_xgb)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.79      0.81     20129
           1       0.98      0.97      0.98     65138
           2       1.00      1.00      1.00     12099
           3       1.00      1.00      1.00      9591
           4       0.83      0.86      0.84     24350
           5       0.98      0.99      0.98     83418

    accuracy                           0.95    214725
   macro avg       0.94      0.93      0.93    214725
weighted avg       0.95      0.95      0.95    214725
```



Lime Explaination

In [36]:
```
from lime.lime_tabular import LimeTabularExplainer
```

In [37]:
```
explainer = LimeTabularExplainer(X_train.values, mode="classification")
```

In [38]:
```
#a specific data point for explaination
data_point = X_test.iloc[0]
```

In [39]:
```
explanation = explainer.explain_instance(data_point.values, xgb.predict_
```

In [40]:
```
explanation.show_in_notebook()
```

SHAP Explaination

In [41]:
```
import shap
```

In [42]:
```
shap.initjs()
```

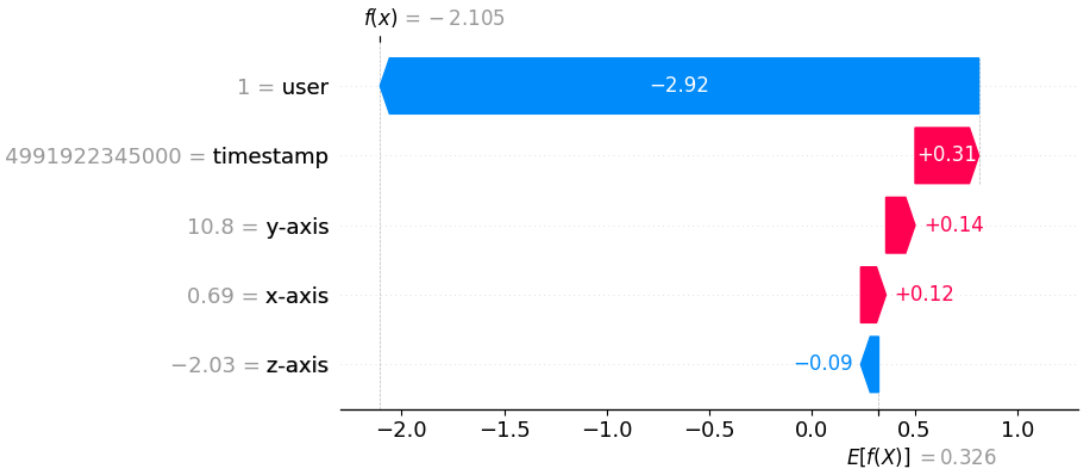
In [44]:
```
explainer = shap.Explainer(xgb)
shap_values = explainer(X1[0:100])
```

```
[22:45:25] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-aut
oscaling-group-i-0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_a
pi\c_api.cc:1240: Saving into deprecated binary model format, please co
nsider using `json` or `ubj`. Model format will default to JSON in XGBo
ost 2.2 if not specified.
[22:45:27] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-aut
oscaling-group-i-0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_a
pi\c_api.cc:1240: Saving into deprecated binary model format, please co
nsider using `json` or `ubj`. Model format will default to JSON in XGBo
ost 2.2 if not specified.
```

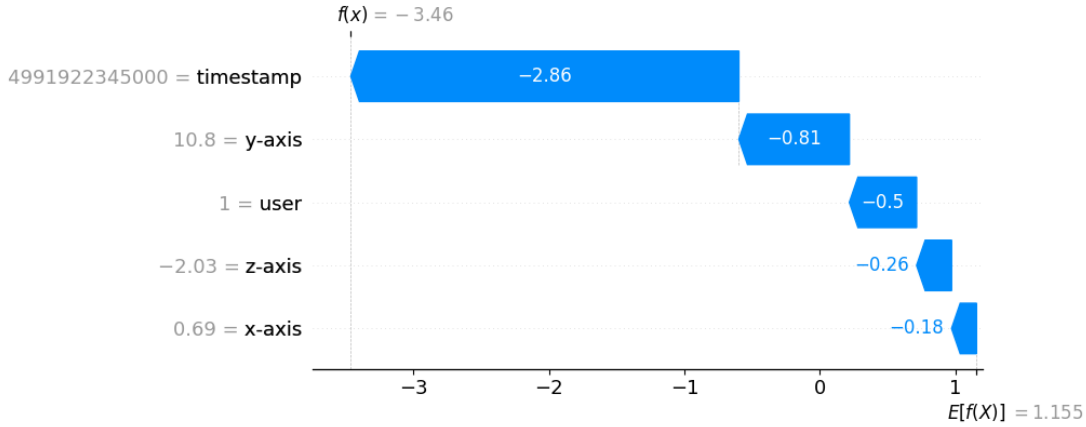
In [45]:
```
np.shape(shap_values.values)
```

Out[45]:
```
(100, 5, 6)
```

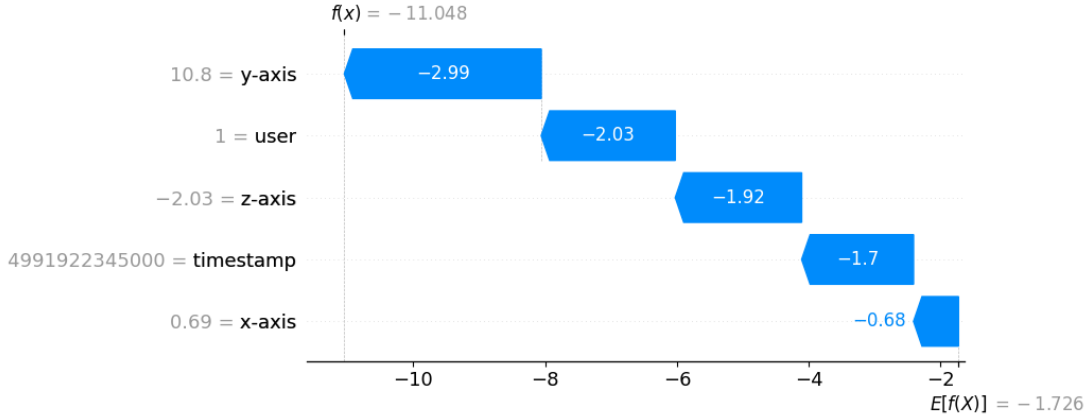
In [46]:
```
shap.plots.waterfall(shap_values[0,:,0])
```



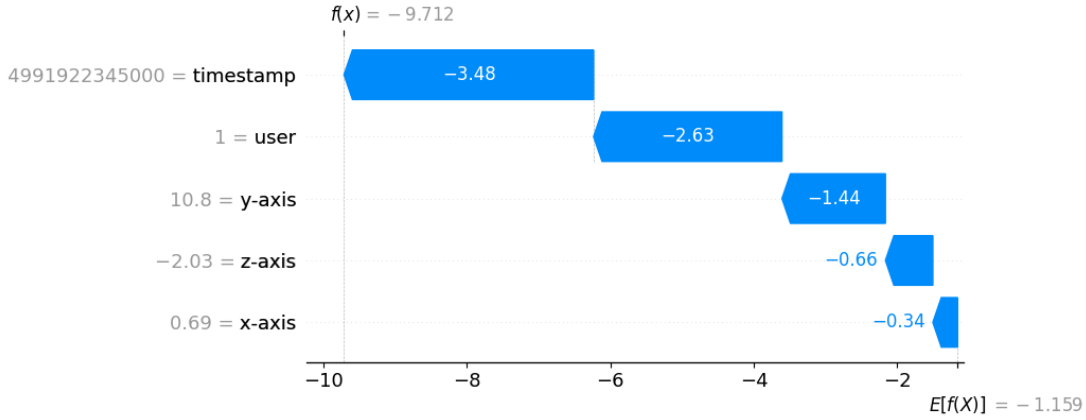
In [47]:
```
shap.plots.waterfall(shap_values[0,:,1])
```



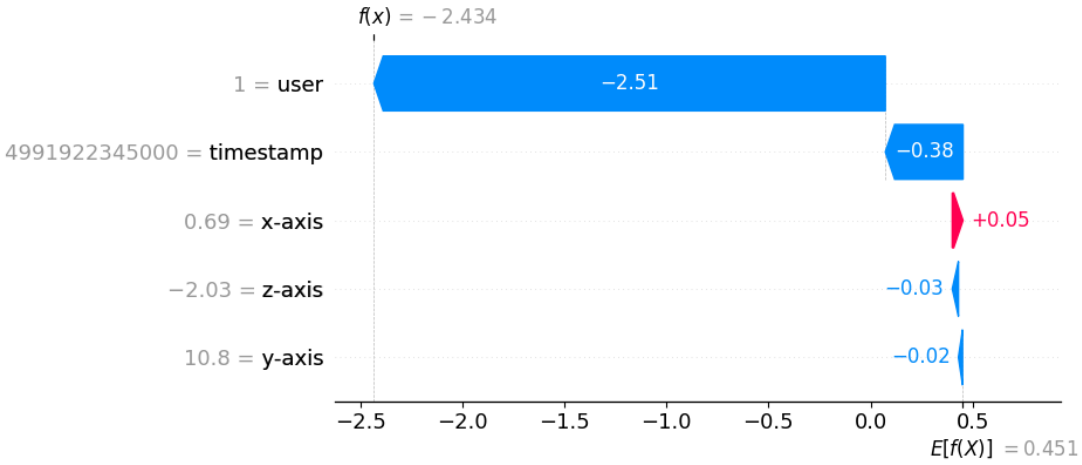
In [48]:
```
shap.plots.waterfall(shap_values[0,:,2])
```



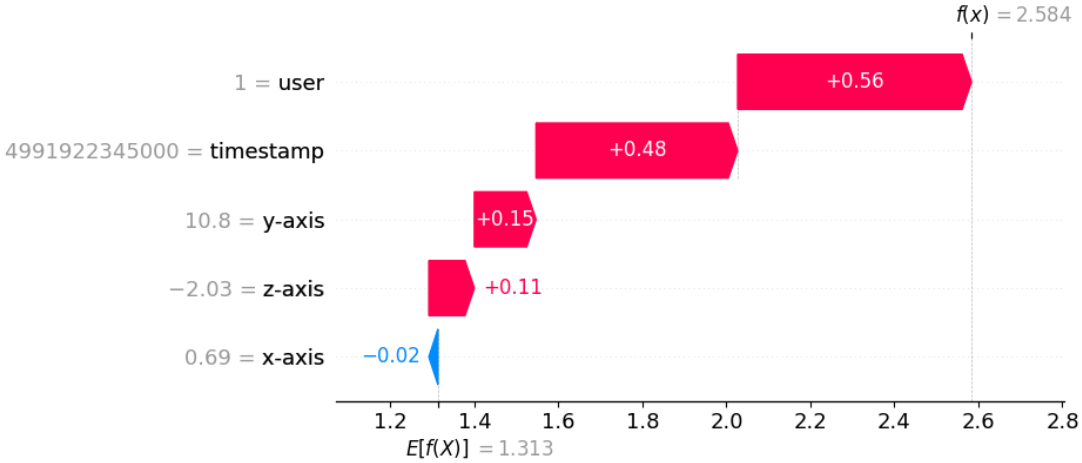
In [49]:
```
shap.plots.waterfall(shap_values[0,:,3])
```



In [50]:
```
shap.plots.waterfall(shap_values[0,:,4])
```



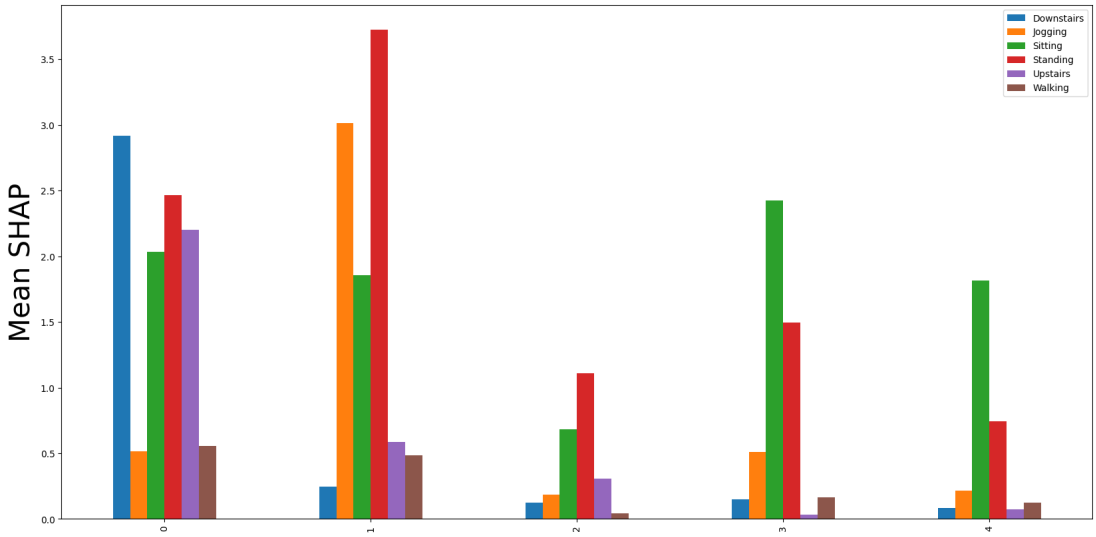
In [51]:
```
shap.plots.waterfall(shap_values[0,:,5])
```



In [52]:

```
#mean shap values calculation for the entire dataset

mean_0 = np.mean(np.abs(shap_values.values[:,:,0]),axis=0)
mean_1 = np.mean(np.abs(shap_values.values[:,:,1]),axis=0)
mean_2 = np.mean(np.abs(shap_values.values[:,:,2]),axis=0)
mean_3 = np.mean(np.abs(shap_values.values[:,:,3]),axis=0)
mean_4 = np.mean(np.abs(shap_values.values[:,:,4]),axis=0)
mean_5 = np.mean(np.abs(shap_values.values[:,:,5]),axis=0)


df1 = pd.DataFrame({'Downstairs':mean_0,'Jogging':mean_1,'Sitting':mean_

# plot mean SHAP values
fig,ax = plt.subplots(1,1,figsize=(20,10))
df1.plot.bar(ax=ax)

ax.set_ylabel('Mean SHAP',size = 30)
ax.set_xticklabels(X.columns,rotation=45,size=20)
ax.legend(fontsize=20)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[52], line 18
     15 df1.plot.bar(ax=ax)
     17 ax.set_ylabel('Mean SHAP',size = 30)
---> 18 ax.set_xticklabels(X.columns,rotation=45,size=20)
     19 ax.legend(fontsize=20)

File ~\AppData\Roaming\Python\Python310\site-packages\matplotlib\axes\_base.py:73, in _axis_method_wrapper.__set_name__.<locals>.wrapper(self, *args, **kwargs)
     72 def wrapper(self, *args, **kwargs):
---> 73     return get_method(self)(*args, **kwargs)

File ~\AppData\Roaming\Python\Python310\site-packages\matplotlib\_api\deprecation.py:297, in rename_parameter.<locals>.wrapper(*args, **kwargs)
    292     warn_deprecated(
    293         since, message=f"The {old!r} parameter of {func.__name__}() "
    294         f"has been renamed {new!r} since Matplotlib {since}; support "
    295         f"for the old name will be dropped %(removal)s.")
    296     kwargs[new] = kwargs.pop(old)
--> 297 return func(*args, **kwargs)

File ~\AppData\Roaming\Python\Python310\site-packages\matplotlib\axis.py:2025, in Axis.set_ticklabels(self, labels, minor, fontdict, **kwargs)
   2021 elif isinstance(locator, mticker.FixedLocator):
   2022     # Passing [] as a list of labels is often used as a way to
   2023     # remove all tick labels, so only error for > 0 labels
   2024     if len(locator.locs) != len(labels) and len(labels) != 0:
-> 2025         raise ValueError(
   2026             "The number of FixedLocator locations"
   2027             f" ({len(locator.locs)}), usually from a call to"
   2028             " set_ticks, does not match"
   2029             f" the number of labels ({len(labels)}).")
   2030     tickd = {loc: lab for loc, lab in zip(locator.locs, labels)}
   2031     func = functools.partial(self._format_with_dict, tickd)

ValueError: The number of FixedLocator locations (5), usually from a call to set_ticks, does not match the number of labels (6).
```



In [ ]:

In [ ]:

K-Nearest Neighbour

In [58]:

```
df3=pd.read_csv('HARR.csv')
df3 = df3.dropna()
df3.head()
```

Out[58]:

	user	activity	timestamp	x-axis	y-axis	z-axis
0	1	Walking	4991922345000	0.69	10.80	-2.03
1	1	Walking	4991972333000	6.85	7.44	-0.50
2	1	Walking	4992022351000	0.93	5.63	-0.50
3	1	Walking	4992072339000	-2.11	5.01	-0.69
4	1	Walking	4992122358000	-4.59	4.29	-1.95

In [59]:

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df3['encoded_activity'] = label_encoder.fit_transform(df3['activity'])

print(df3['encoded_activity'].unique())
```

```
[5 1 4 0 2 3]
```

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

X3 = df3.drop(['activity', 'encoded_activity'], axis=1)
y3 = df3['encoded_activity']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

In [61]:
```
from sklearn.preprocessing import MinMaxScaler

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

In [62]:
```
from sklearn.neighbors import KNeighborsClassifier



knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train_scaled, y_train)

y_pred_knn = knn.predict(X_test_scaled)

accuracy_knn = knn.score(X_test_scaled, y_test)
print(f"Accuracy of KNN: {accuracy_knn}")
```

```
Accuracy of KNN: 0.7893398533007335
```

In [63]:
```
print(f"Accuracy: {accuracy_knn * 100:.2f}%")
```

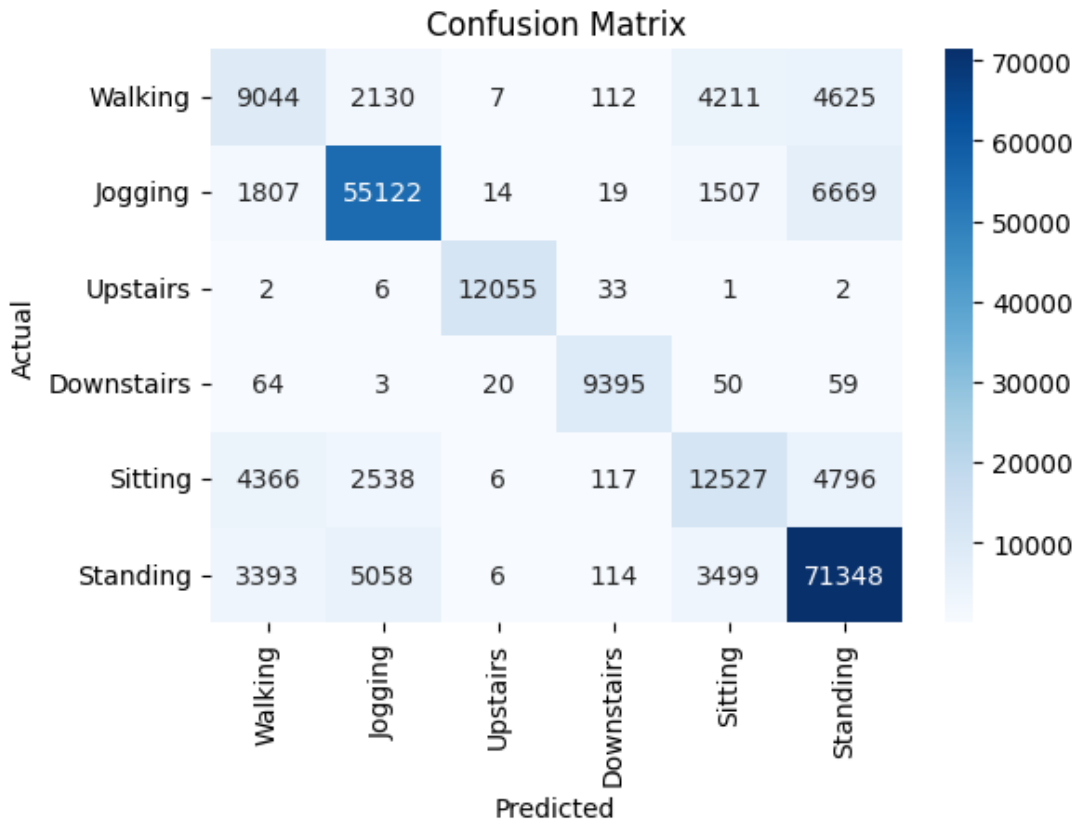
```
Accuracy: 78.93%
```

In [64]:
```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

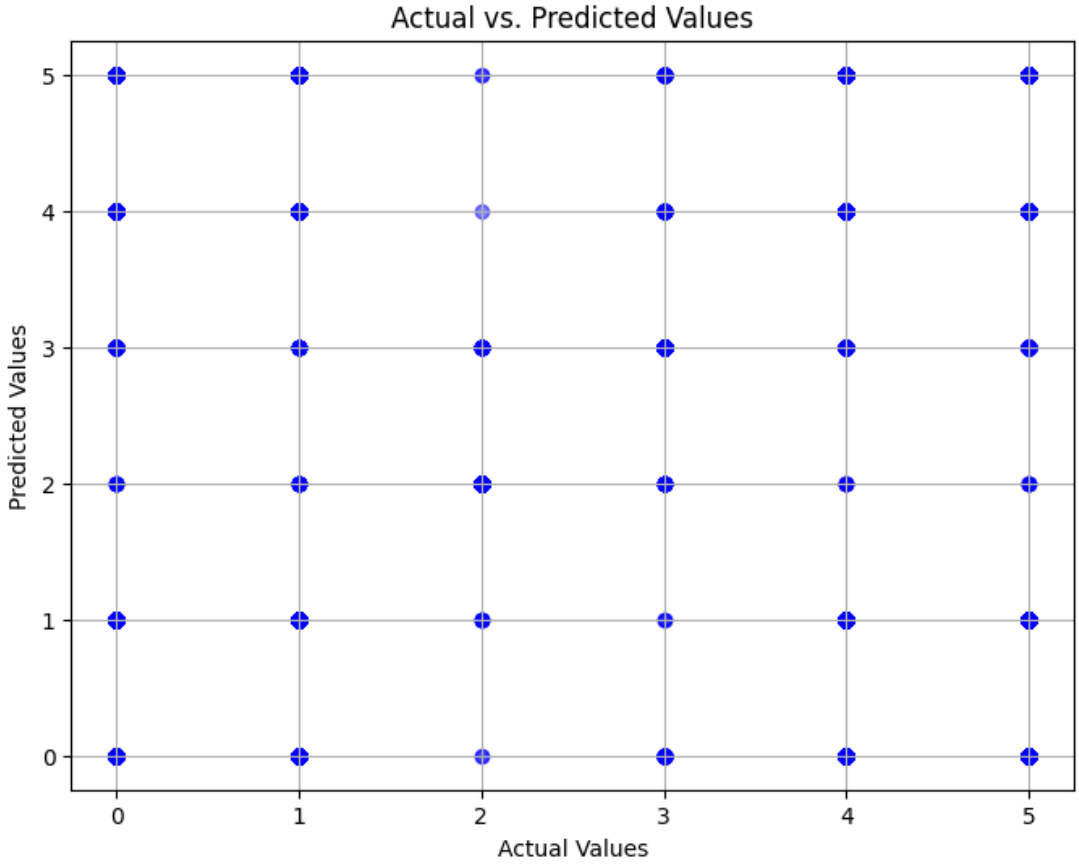
In [65]:

```
# Classification Report
print("Classification Report:\n", classification_report(y_test, y_pred_k

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.48      0.45      0.47     20129
           1       0.85      0.85      0.85     65138
           2       1.00      1.00      1.00     12099
           3       0.96      0.98      0.97      9591
           4       0.57      0.51      0.54     24350
           5       0.82      0.86      0.83     83418

    accuracy                           0.79    214725
   macro avg       0.78      0.77      0.78    214725
weighted avg       0.78      0.79      0.79    214725
```



In [66]:
```
import matplotlib.pyplot as plt
import numpy as np


plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_knn, color='b', alpha=0.5)
plt.title('Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.grid(True)
plt.show()
```



In [ ]:

In [ ]:

Logistic Regressor

In [67]:
```
df4=pd.read_csv('har.csv')
df4 = df4.dropna()
df4.head()
```

Out[67]:

	user	activity	timestamp	x-axis	y-axis	z-axis
0	1	Walking	4.991920e+12	0.69	10.80	-2.03
1	1	Walking	4.991970e+12	6.85	7.44	-0.50
2	1	Walking	4.992020e+12	0.93	5.63	-0.50
3	1	Walking	4.992070e+12	-2.11	5.01	-0.69
4	1	Walking	4.992120e+12	-4.59	4.29	-1.95

In [68]:
```
missing_values = df4.isnull().sum()
print("Missing Values:\n", missing_values)
```

```
Missing Values:
 user         0
activity     0
timestamp    0
x-axis       0
y-axis       0
z-axis       0
dtype: int64
```

In [69]:
```
label_encoder = LabelEncoder()
df4['activity'] = label_encoder.fit_transform(df4['activity'])
```

In [70]:
```
# Step 3: Feature Scaling
```

In [71]:
```
from sklearn.preprocessing import StandardScaler, LabelEncoder

scaler = StandardScaler()
df4[['x-axis', 'y-axis', 'z-axis']] = scaler.fit_transform(df4[['x-axis'
```

In [72]:
```
df4['timestamp'] = pd.to_datetime(df4['timestamp'], unit='ns')  # Change

df4['hour_of_day'] = df4['timestamp'].dt.hour
df4['day_of_week'] = df4['timestamp'].dt.dayofweek

df4 = df4.drop('timestamp', axis=1)

print("Modified DataFrame:")
print(df4.head())
```

```
Modified DataFrame:
   user  activity    x-axis    y-axis    z-axis  hour_of_day  day_of_we
ek
0     1         5  0.025476  0.538037 -0.484426            1
3
1     1         5  0.889636  0.031366 -0.173546            1
3
2     1         5  0.059145 -0.241573 -0.173546            1
3
3     1         5 -0.367324 -0.335066 -0.212152            1
3
4     1         5 -0.715232 -0.443639 -0.468171            1
3
```

In [73]:
```
from sklearn.utils.class_weight import compute_class_weight
from sklearn.utils import resample
```

In [74]:
```

class_weights = compute_class_weight('balanced', classes=df4['activity']
class_weight_dict = dict(zip(df4['activity'].unique(), class_weights))

class_dfs = [df4[df4['activity'] == cls] for cls in df4['activity'].uniq

df_upsampled = pd.concat([resample(class_df4, replace=True, n_samples=le
                          for class_df4 in class_dfs])
```

In [75]:
```
X_train, X_test, y_train, y_test = train_test_split(df4.drop('activity',
```

In [76]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

In [77]:
```
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)
```

Out[77]:
```
LogisticRegression(max_iter=1000)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [78]:
```
y_pred_lr = logistic_model.predict(X_test)
```

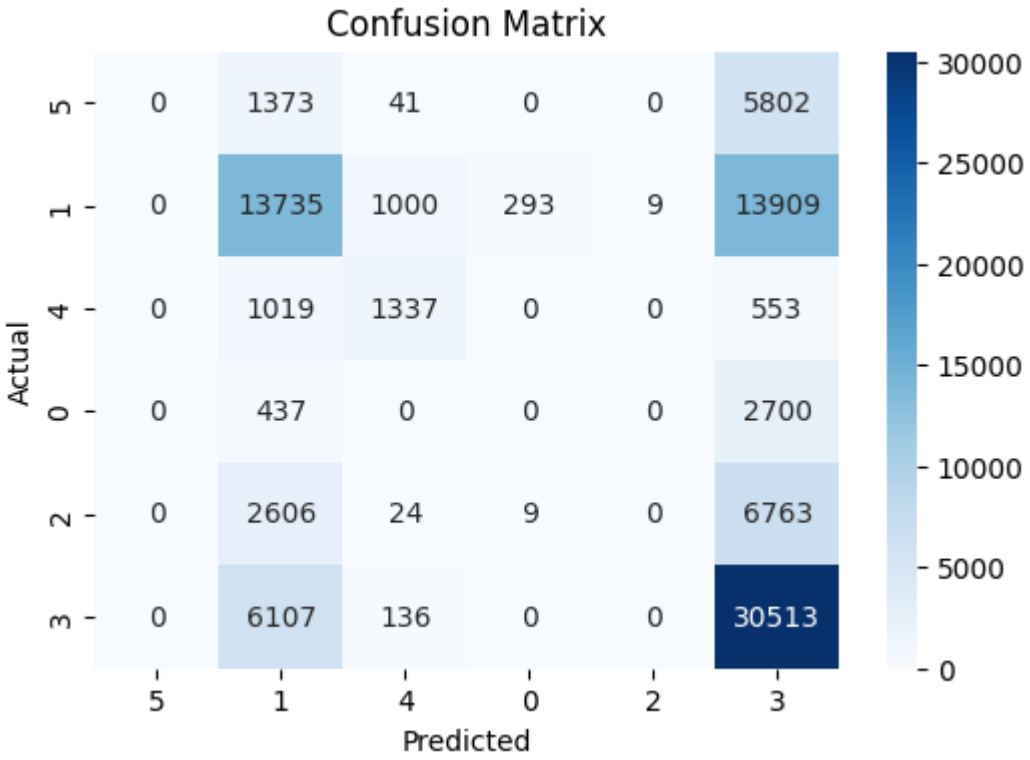
In [79]:
```
accuracy_lr = logistic_model.score(X_test, y_test)
print(f"Accuracy of lr: {accuracy_lr}")
```

```
Accuracy of lr: 0.5158658307493832
```

In [80]:

```
# Classification Report
print("Classification Report:\n", classification_report(y_test, y_pred_l

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_lr)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

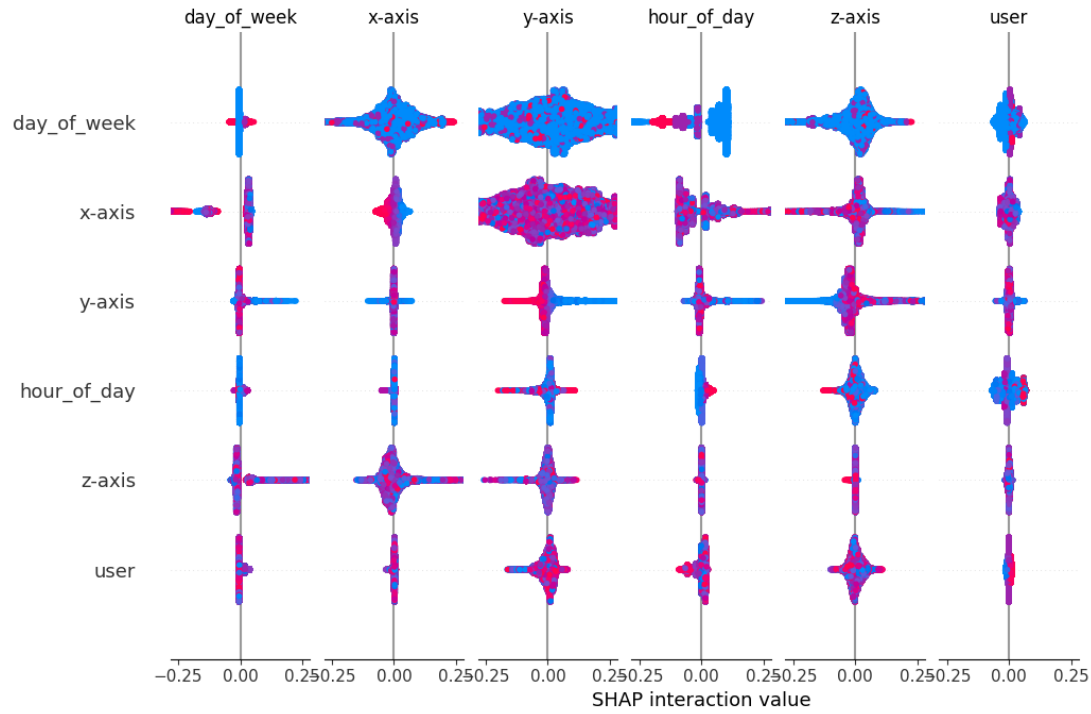
```
Precision and F-score are ill-defined and being set to 0.0 in labels wi
th no predicted samples. Use `zero_division` parameter to control this
behavior.
Precision and F-score are ill-defined and being set to 0.0 in labels wi
th no predicted samples. Use `zero_division` parameter to control this
behavior.
Precision and F-score are ill-defined and being set to 0.0 in labels wi
th no predicted samples. Use `zero_division` parameter to control this
behavior.

Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00      7216
           1       0.54      0.47      0.51     28946
           2       0.53      0.46      0.49      2909
           3       0.00      0.00      0.00      3137
           4       0.00      0.00      0.00      9402
           5       0.51      0.83      0.63     36756

    accuracy                           0.52     88366
   macro avg       0.26      0.29      0.27     88366
weighted avg       0.41      0.52      0.44     88366
```


Confusion Matrix
Actual
Predicted
5
1
4
0
2
3
0 1373 41 0 0 5802
0 13735 1000 293 9 13909
0 1019 1337 0 0 553
0 437 0 0 0 2700
0 2606 24 9 0 6763
0 6107 136 0 0 30513
30000
25000
20000
15000
10000
5000
0

In [81]:

```
import shap

#Explain Model Predictions using SHAP

logistic_predictor = lambda x: logistic_model.predict_proba(x)

explainer = shap.Explainer(logistic_predictor, X_train)

shap_values = explainer(X_test)

shap.summary_plot(shap_values, X_test, feature_names=X_test.columns, cla

shap.force_plot(explainer.expected_value[0], shap_values[0][0, :], X_tes
```

```
ExactExplainer explainer: 88367it [11:47, 124.89it/s]
```

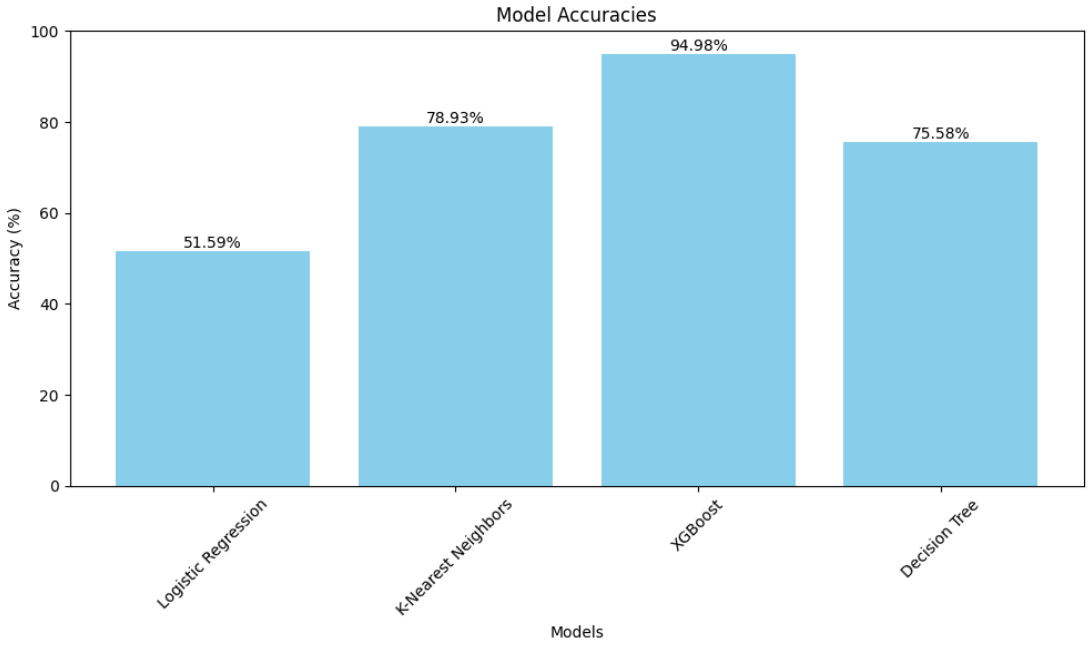



```
---------------------------------------------------------------------
----
AttributeError                            Traceback (most recent call l
ast)
Cell In[81], line 13
      9 shap_values = explainer(X_test)
     11 shap.summary_plot(shap_values, X_test, feature_names=X_test.col
umns, class_names=logistic_model.classes_)
---> 13 shap.force_plot(explainer.expected_value[0], shap_values[0][0,
:], X_test.iloc[0, :], feature_names=X_test.columns)

AttributeError: 'ExactExplainer' object has no attribute 'expected_valu
e'
```

In [82]:
```
print(f"Accuracy: {accuracy_lr * 100:.2f}%")
print(f"Accuracy: {accuracy_knn * 100:.2f}%")
print(f"Accuracy: {accuracy_xgb * 100:.2f}%")
print(f"Accuracy: {accuracy_dt * 100:.2f}%")
```

```
Accuracy: 51.59%
Accuracy: 78.93%
Accuracy: 94.98%
Accuracy: 75.58%
```

In [83]:
```
accuracies = [accuracy_lr, accuracy_knn, accuracy_xgb, accuracy_dt]
model_names = ['Logistic Regression', 'K-Nearest Neighbors', 'XGBoost',


plt.figure(figsize=(10, 6))

plt.bar(model_names, [acc * 100 for acc in accuracies], color='skyblue')
plt.xlabel('Models')
plt.ylabel('Accuracy (%)')
plt.title('Model Accuracies')

for i in range(len(model_names)):
    plt.text(i, accuracies[i] * 100, f"{accuracies[i] * 100:.2f}%", ha='

plt.ylim(0, 100)

plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



In [ ]: