Bicycles

November 4, 2025

[ ]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import copy
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from sklearn.metrics import classification_report
from sklearn.linear_model import LinearRegression
```

[ ]:
```
#Data Cleaning and Labelling
dataset_cols=["bike_count","hour","temp","humidity","wind","visibility","dew_pt_temp","radiati
df=pd.read_csv("SeoulBikeData.csv",encoding='cp949').
 ↪drop(["Date","Holiday","Seasons"], axis=1)
df.columns=dataset_cols
df["functional"]==(df["functional"]=="Yes").astype(int)
df=df[df["hour"]==12]
df=df.drop(["hour"],axis=1)
```

[ ]:
```
#Data Plotting
for label in df.columns[1:]:
    plt.scatter(df[label],df["bike_count"])
    plt.title(label)
    plt.xlabel(label)
    plt.ylabel("Bike count at noon")
    plt.show()
```

[ ]:
```
df=df.drop(["wind","visibility","functional"],axis=1)
df.head()
```

[ ]:
```
#Train/valid/test datasets
train,val,test=np.split(df.sample(frac=1),[int(0.6*len(df)),int(0.8*len(df))])
def get_xy(dataframe,y_label,x_labels=None):
    if not x_labels:
        x=dataframe[[c for c in dataframe.columns if c!=y_label]].values
    else:
```

```
        if len(x_labels)==1:
            X=dataframe[x_labels[0]].values.reshape(-1,1)
        else:
            X=dataframe[x_labels].values


    y=dataframe[y_label].values.reshape(-1,1)
    data=np.hstack((X,y))

    return data,X,y

_,X_train_temp,y_train_temp=get_xy(train,"bike_count",x_labels=["temp"])
_,X_val_temp,y_val_temp=get_xy(val,"bike_count",x_labels=["temp"])
_,X_test_temp,y_test_temp=get_xy(test,"bike_count",x_labels=["temp"])
```

[ ]:
```
#Traning Linear Regression Model
temp_reg=LinearRegression()
temp_reg.fit(X_train_temp,y_train_temp)
temp_reg.score(X_test_temp,y_test_temp)
```

[ ]:
```
plt.scatter(X_train_temp,y_train_temp,label="Data",color="Blue")
x=tf.linspace(-20,40,200)
plt.plot(x,temp_reg.predict(np.array(x).
 ↪reshape(-1,1)),label="Fit",color="red",linewidth=3)
plt.title("Bikes vrs Temp")
plt.ylabel("Number of Bikes")
plt.xlabel("Temp")
plt.legend()
plt.show()
```

[ ]:
```
def plot_loss(history):
    plt.plot(history.history['loss'], label='loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.xlabel('Epoch')
    plt.ylabel("MSE")
    plt.legend()
    plt.grid(True)
    plt.show()
```

[ ]:
```
#Regression Training with Neural Network
temp_normalizer=tf.keras.layers.Normalization(input_shape=(1,),axis=None)
temp_normalizer.adapt(X_train_temp.reshape(-1))
temp_nn_model=tf.keras.Sequential([
    temp_normalizer,
    tf.keras.layers.Dense(1)
])
```

```
temp_nn_model.compile(optimizer=tf.keras.optimizers.Adam(0.
  ↪1),loss="mean_squared_error")
history=temp_nn_model.fit(
    X_train_temp.reshape(-1),y_train_temp,
    epochs=1000,
    validation_data=(X_val_temp,y_val_temp),
    verbose=0
)
```

[ ]:
```
plot_loss(history)

#Model acccuracy before NN implementation
plt.scatter(X_train_temp,y_train_temp,label="Data",color="Blue")
x=tf.linspace(-20,40,200)
plt.plot(x,temp_reg.predict(np.array(x).
  ↪reshape(-1,1)),label="Fit",color="red",linewidth=3)
plt.title("Model before NN implementation")
plt.ylabel("Number of Bikes")
plt.xlabel("Temp")
plt.legend()
plt.show()

#Model acccuracy after NN implementation
plt.scatter(X_train_temp,y_train_temp,label="Data",color="Blue")
x=tf.linspace(-20,40,200)
plt.plot(x,temp_nn_model.predict(np.array(x).
  ↪reshape(-1,1)),label="Fit",color="green",linewidth=3)
plt.title("Model after NN implementation")
plt.ylabel("Number of Bikes")
plt.xlabel("Temp")
plt.legend()
plt.show()
```