# Pseudocode for IS 160 A1 Sim & Ojas

- Import libraries
- Load CSV files
- Handle missing values & values with zero's
- Define the Euclidean distance function
- Calculate the distance between pickup & dropoff
- Drop redundant tables (pickup-date time & pickup+dropoff tables)
- Reorder columns
- Print data frame to see seperated pickup time & distance
- Define & Manually split Train & Test sets
- Normalize the Data
- Create function to build model
- Build model
- Split Train & Test sets
- Train model
- Extract Training & validation loss for Epoch
- Evaluate model on test data

Writing Code IS 160 A1 Sim & Ojas

```
import pandas as pd
import numpy as np

df = pd.read_csv('bayarea_taxi.csv') #loading up the csv file

df.dropna(inplace=True) #handling the missing values

df = df[(df['pickup_longitude'] != 0) & (df['pickup_latitude'] != 0) & (df['dropoff_longitude'] != 0) & (df['dropoff_latitude'] != 0)]
                                    #dropping rows that had 0's

#Extracting the components of pickup_datetime
df['year'] = df['pickup_datetime'].dt.year
df['month'] = df['pickup_datetime'].dt.month
df['day'] = df['pickup_datetime'].dt.day
df['day of week'] = df['pickup_datetime'].dt.day of week
df['hour'] = df['pickup_datetime'].dt.hour
df['minute'] = df['pickup_datetime'].dt.minute
df['second'] = df['pickup_datetime'].dt.second

#Defining the Euclidean distance function
def euc_distance(lat1, long1, lat2, lat2):
    return((lat1-lat2)**2+(long1-long2)**2)**0.5)

#Calculating the distance between pickup and dropoff
df['distance'] = euc_distance(df['pickup_latitude'],
                              df['pickup_longitude'],
                              df['dropoff_latitude'],
                              df['dropoff_longitude'])
```

```
# Dropping pickup_datetime & pickup & dropoff
since we seperated its components
df = df.drop(columns = ['pickup_datetime',])
df = df.drop(columns = ['pickup_latitude',])
df = df.drop(columns = ['pickup_longitude',])
df = df.drop(columns = ['dropoff_latitude',])
df = df.drop(columns = ['dropoff_longitude',])

# Reordering columns & reassigning data frame
cols = df.columns.tolist()

cols.remove('distance')
distance_index = cols.index('passanger')+1
cols.insert(distance_index, 'distance')

df = df[cols]

#Printing dataframe to see each seperated and calculated distance
print(df.loc[:24,:])

# Defining the new input features & splitting manually into train & test sets
train_data = df[['passanger', 'distance', 'year', 'month', 'day',
'day_of_week', 'hour']]

train_targets = df['fare_amount']

from sklearn.model_selection import train_test_split
train_data, test_data, train_targets, test_targets =
train_test_split(train_data, train_targets, test_size =
0.2, random_state=42)

train_data[:]
train_targets[:]
```

```
#Normalizing the data (mean normalization)
mean = train_data.mean(axis=0)
std = train_data.std(axis=0)
train_data = (train_data - mean) / std
test_data = (test_data - mean) / std

#Function to build the model
def build_model():
    model = models.Sequential()

    model.add(layers.Dense(64, activation="relu", input_
shape=(train_data.shape[1],)))

    model.add(layers.Dense(64, activation="relu"))

    model.add(layers.Dense(1))

    model.compile(optimizer='rmsprop', loss='mse', metrics=
["mae"])

    return model
```

```
#Building the model
model = build_model()

#Splitting training data further into partial training &
validation data
partial_train_data = train_data[:int(0.8*len(train_data))]
partial_train_targets = train_targets[:int(0.8*len(train_targets))]
val_data = train_data[int(0.8*len(train_data)):]
val_targets = train_targets[int(0.8*len(train_targets)):]

#Training the model
num_epochs = 20
history = model.fit(
    partial_train_data,
    partial_train_targets,
    validation_data = (val_data, val_targets),
    epochs = num_epochs,
    batch_size = 1,
    verbose = 1
)

#Extracting the training & validation loss for epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

#Evaluating the model on test data
test_mse_score, test_mae_score = model.evaluate(test_data,
test_targets)
print(f"Test MAE: {test_mae_score}")
```