Here's the implementation for the tasks described using the **Taxi Demand Prediction in New York City** dataset. The code covers all stages from Week 1-4: business problem definition, data cleaning, feature engineering, modeling, and evaluation.

Step 1: Business Problem Overview

- **Objective**: Predict taxi demand in New York City using historical data.
- **Constraints**:
  - Handle large data efficiently.
  - Ensure the model is interpretable.
  - Provide time-sensitive predictions.

Step 2: Mapping to DS Problem

Load and Inspect Data

```
# Import required libraries
import dask.dataframe as dd

# Load NYC taxi data (replace with the actual dataset path)
data = dd.read_csv("yellow_tripdata_2015-01.csv")

# Display basic information
print(data.head())
print(data.info())
```

Step 3: Data Cleaning

Clean Latitude and Longitude Data

```
# Assuming columns 'PULocationID' and 'DOLocationID' represent coordinates
data = data[(data['PULocationID'] ⩾ 1) & (data['DOLocationID'] ⩾ 1)]
```

Clean Trip Duration

```
data['trip_duration'] = (dd.to_datetime(data['tpep_dropoff_datetime']) -

dd.to_datetime(data['tpep_pickup_datetime'])).dt.total_seconds()

# Remove invalid trip durations (e.g., <1 minute or >6 hours)
data = data[(data['trip_duration'] > 60) & (data['trip_duration'] ⩽ 21600)]
```

Clean Speed

```
data['speed'] = data['trip_distance'] / (data['trip_duration'] / 3600)
data = data[(data['speed'] > 0) & (data['speed'] ⩽ 100)]
```

Clean Distance

```
# Remove unrealistic trip distances
data = data[(data['trip_distance'] > 0) & (data['trip_distance'] ⩽ 100)]
```

Clean Fare

```
# Remove invalid fare values
data = data[(data['fare_amount'] > 0) & (data['fare_amount'] ⩽ 500)]
```

Remove Outliers

```
# Remove remaining outliers
data = data[(data['trip_distance'] < 50) & (data['speed'] < 100) &
(data['fare_amount'] < 500)]
```

Step 4: Data Preparation

Clustering and Segmentation

```
from sklearn.cluster import KMeans
import pandas as pd

# Convert Dask dataframe to Pandas for clustering
data_pd = data.compute()
kmeans = KMeans(n_clusters=30, random_state=42)
data_pd['pickup_cluster'] = kmeans.fit_predict(data_pd[['PULocationID',
'DOLocationID']])
```

Time Binning

```
data_pd['time_bin'] =
pd.to_datetime(data_pd['tpep_pickup_datetime']).dt.floor('10min')
```

Smoothing Time-Series Data

```
# Group by time bin and cluster, and fill missing values
data_pd['pickup_count'] = data_pd.groupby(['pickup_cluster', 'time_bin'])
['trip_distance'].transform('count')
data_pd['pickup_count'] = data_pd['pickup_count'].fillna(0)
```

Fourier Transform

```
from scipy.fftpack import fft
import numpy as np

# Fourier Transform of pickup count
data_pd['fft_pickup'] = np.abs(fft(data_pd['pickup_count']))
```

Step 5: Moving Averages

Simple Moving Average

```
data_pd['sma'] = data_pd['pickup_count'].rolling(window=5).mean()
```

Weighted Moving Average

```
weights = np.arange(1, 6)
data_pd['wma'] = data_pd['pickup_count'].rolling(window=5).apply(
    lambda x: np.dot(x, weights) / weights.sum(), raw=True)
```

Exponential Weighted Moving Average

```
data_pd['ewma'] = data_pd['pickup_count'].ewm(span=5, adjust=False).mean()
```

Step 6: Train-Test Split and Feature Engineering

```
from sklearn.model_selection import train_test_split

# Select features and target
features = ['pickup_cluster', 'sma', 'wma', 'ewma', 'fft_pickup',
'trip_distance', 'speed']
target = 'pickup_count'

X = data_pd[features]
y = data_pd[target]

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Step 7: Regression Models

Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

print("Linear Regression MSE:", mean_squared_error(y_test, y_pred_lr))
```

Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

print("Random Forest MSE:", mean_squared_error(y_test, y_pred_rf))
```

XGBoost Regression

```
import xgboost as xgb

xg = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,
random_state=42)
xg.fit(X_train, y_train)
y_pred_xg = xg.predict(X_test)
```

```
print("XGBoost MSE:", mean_squared_error(y_test, y_pred_xg))
```

Step 8: Results and Model Comparison

```
results = pd.DataFrame({
    "Model": ["Linear Regression", "Random Forest", "XGBoost"],
    "MSE": [
        mean_squared_error(y_test, y_pred_lr),
        mean_squared_error(y_test, y_pred_rf),
        mean_squared_error(y_test, y_pred_xg)
    ]
})
print(results)
```

Modified MAPE

```
def modified_mape(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / (y_true + 1))) * 100  # +1 to
avoid division by zero

print("Linear Regression Modified MAPE:", modified_mape(y_test, y_pred_lr))
print("Random Forest Modified MAPE:", modified_mape(y_test, y_pred_rf))
print("XGBoost Modified MAPE:", modified_mape(y_test, y_pred_xg))
```

This implementation completes all tasks from Week 1-4. Let me know if you need additional enhancements or explanations!