Main.py

```python
import pandas as pd
import numpy as np

# Define paths for each dataset in your Google Drive
df1_path = '/Users/ankushbhatt/Downloads/VFS-CS/DataSet/groundtruth.csv'  # Update with your actual path
df2_path = '/Users/ankushbhatt/Downloads/VFS-CS/DataSet/weather_features.csv'  # Update with your actual path
df3_path = '/Users/ankushbhatt/Downloads/VFS-CS/DataSet/road_features.csv'  # Update with your actual path

# define the datatypes for each csv
dtypes_df1 = {'road_segment_id': 'int32', 'timestamp': 'str', 'max_capacity': 'int32', 'occupied': 'int32', 'available': 'int32'}
dtypes_df2 = {'road_segment_id': 'int32', 'timestamp': 'str', 'tempC': 'float32', 'windspeedKmph': 'float32', 'precipMM': 'float32'}
dtypes_df3 = {'road_segment_id': 'int32', 'commercial': 'Int64', 'residential': 'Int64', 'transportation': 'Int64', 'schools': 'Int64', 'eventsites': 'Int64',
              'restaurant': 'Int64', 'shopping': 'Int64', 'office': 'Int64', 'supermarket': 'Int64', 'num_off_street_parking': 'Int64', 'off_street_capa': 'Int64'}

# load the datasets using the defined paths
df1 = pd.read_csv(df1_path, dtype=dtypes_df1)
df2 = pd.read_csv(df2_path, dtype=dtypes_df2)
df3 = pd.read_csv(df3_path, dtype=dtypes_df3)


# merge the dataset
# first merge df1 and df2 on road_segment_id and timestamp
merged_df = pd.merge(df1, df2, on=['road_segment_id', 'timestamp'], how='inner')
# second merge df3 with merged_df on road_segment_id
final_df = pd.merge(merged_df, df3, on='road_segment_id', how='inner')

# save the merged dataset
final_df.to_csv('New_joint_dataset.csv', index=False)

# Print the final merged dataset info
print("\nMerged Dataset:")
print(final_df.info()) # showing information about final merged dataset
print(final_df.head()) # showing first few roows of finam merged dataset

# Data Pre processing

# Load New Joint dataset

import pandas as pd
import numpy as np
df = pd.read_csv('New_joint_dataset.csv')

# get a summary of the data
print("First few rows of dataframe")
print(df.head())
print("\ndataframe info:")
print(df.info())
print("\nsummary of dataframe:")
print(df.describe(include='all'))

# Check for missing values
missing_values = df.isnull().sum()
print("\nMissing Values:")
print(missing_values)

# Fill the missing values
# Now fill numeric values with median and categorical with mode
for column in final_df.columns:
  if df[column].dtype in ['float64', 'int64']:
    df[column].fillna(df[column].median(), inplace=True)
  elif df[column].dtype == 'object':
    df[column].fillna(df[column].mode()[0], inplace=True)

# Identify and handle the outliers
# Now using IQR methods for detecting outliers in the data
def remove_outliers_iqr(data):
  for column in data.select_dtypes(include=['float64', 'int64']).columns:
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data = data[(data[column] >= lower_bound) & (data[column] <= upper_bound)]
  return data
df = remove_outliers_iqr(df)

# Standardize the format of data
df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')  # 'coerce' will turn invalid parsing into NaT (Not a Time)

# Extract useful date-time features
df['year'] = df['timestamp'].dt.year
df['month'] = df['timestamp'].dt.month
df['day'] = df['timestamp'].dt.day
df['hour'] = df['timestamp'].dt.hour
df['minute'] = df['timestamp'].dt.minute
df['dayofweek'] = df['timestamp'].dt.dayofweek  # Monday=0, Sunday=6

# Drop the original 'timestamp' column as it is no longer needed for prediction
df.drop(columns=['timestamp'], inplace=True)

# Now you can proceed with other steps (e.g., missing values handling, outliers removal, etc.)

# Now removing duplicates
df.drop_duplicates(inplace=True)

# dropping the unnecessary columns
```

```python
df.drop(columns=['Unnamed: 0_x', 'Unnamed: 0_y', 'Unnamed: 0', 'max_capacity', 'occupied'], inplace=True, errors='ignore')
print("Columns after dropping:", df.columns)

# convert data types
# convert numeric columns to appropriate types
from sklearn.preprocessing import StandardScaler, PowerTransformer
numeric_columns = ['max_capacity', 'occupied', 'available', 'tempC', 'windspeedKmph', 'precipMM', 'commercial', 'residential', 'transportation', 'schools', 'eventsites', 'restaurant', 'shopping',
                   'office', 'supermarket', 'num_off_street_parking', 'off_street_capa']
for column in numeric_columns:
  if column in df.columns:
    df[column] = pd.to_numeric(df[column], errors='coerce')

# convert 'road_segment_id' to int32 if necesaary
if 'road_segment_id' in df.columns:
  df['road_segment_id'] = df['road_segment_id'].astype('int32')

#**Skewness Correction with Power Transformation**
# Apply PowerTransformer to correct skewness in numeric columns
numeric_features = df.select_dtypes(include=['float64', 'int64']).columns
power_transformer = PowerTransformer(method='yeo-johnson', standardize=True)
df[numeric_features] = power_transformer.fit_transform(df[numeric_features])

# **Scaling Features**
# Use StandardScaler to standardize numeric features
scaler = StandardScaler()
df[numeric_features] = scaler.fit_transform(df[numeric_features])

# check datatypes after conversion
print("\nData Types after Conversion:")
print(df.dtypes)

# check the info of the final dataframe
print("\nFinal Dataframe Info:")
print(df.info())

# save the clean data
clean_data_path = 'clean_data.csv'
df.to_csv(clean_data_path, index=False)
print("\nData preprocessing completed. clean data saved to", clean_data_path)

# Trained the Model:
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt

# Load the cleaned dataset
```

```python
cd = pd.read_csv('clean_data.csv')

# Features and Target
x = cd.drop(columns=['available'])  # Features
y = cd['available']  # Target

# Split the data into train and test sets (80% train, 20% test)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Define the model with slightly fewer trees and shallower depth
model = RandomForestRegressor(
    n_estimators=30,            # Reduced n_estimators for faster learning
    max_depth=5,                # Limiting tree depth to avoid overfitting
    min_samples_split=5,        # Minimum samples required to split a node
    min_samples_leaf=4,         # Minimum samples required to be in a leaf node
    random_state=42,
    n_jobs=-1                    # Use all available CPU cores for faster training
)

# Fit the model on training data
model.fit(x_train, y_train)

# Predict on test data
y_pred = model.predict(x_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2_score = model.score(x_test, y_test)

# Print evaluation metrics
print("Model Evaluation Metrics:")
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):", rmse)
print("R² (Coefficient of Determination) accuracy:", r2_score)

# Cross-validation with fewer folds (3-fold to save time)
cv_scores = cross_val_score(model, x, y, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
print("\nCross-validation scores:", cv_scores)
print("Average Cross-validation score:", cv_scores.mean())

# Hyperparameter tuning with a focused parameter grid and reduced combinations
param_distributions = {
    'n_estimators': [10, 30, 50],  # Reduced range for faster tuning
    'max_depth': [5, 7],           # Limiting to fewer options
    'min_samples_split': [2, 5],   # Narrowing range
    'min_samples_leaf': [1, 2, 4], # Fewer leaf node options
}

# RandomizedSearchCV with fewer iterations
randomized_search = RandomizedSearchCV(
    RandomForestRegressor(random_state=42, n_jobs=-1),
    param_distributions,
    n_iter=5,                      # Fewer combinations to try
    cv=5,                          # Reduced folds
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

# Fit RandomizedSearchCV to training data
randomized_search.fit(x_train, y_train)

# Print best hyperparameters from RandomizedSearchCV
print("Best Hyperparameters from RandomizedSearchCV:")
print(randomized_search.best_params_)

# Retrain the model with the best parameters
best_model = randomized_search.best_estimator_
best_model.fit(x_train, y_train)

# Predict and evaluate the retrained model
y_pred_best = best_model.predict(x_test)

# Evaluate the model
mae_best = mean_absolute_error(y_test, y_pred_best)
rmse_best = np.sqrt(mean_squared_error(y_test, y_pred_best))
r2_score_best = best_model.score(x_test, y_test)

# Print evaluation metrics of the best model
print("\nBest Model Evaluation Metrics:")
print("Mean Absolute Error (MAE):", mae_best)
print("Root Mean Squared Error (RMSE):", rmse_best)
print("R² (Coefficient of Determination) accuracy:", r2_score_best)

# Final model accuracy
accuracy_percentage_best = r2_score_best * 100
print(f"Final Accuracy: {accuracy_percentage_best:.2f}%")

# Cross-validation to evaluate model performance on multiple folds (using 3-fold CV)
cv_scores_best = cross_val_score(best_model, x, y, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
print("\nCross-validation scores of best model:", cv_scores_best)
print("Average Cross-validation score of best model:", cv_scores_best.mean())

```