Weather Model Demand - ZIP Contents

File: weather_demand_model (1).ipynb


```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "a2dbc77a",
   "metadata": {},
   "source": [
    "# Weather-Driven Electricity Demand Prediction\n",
    "Detailed notebook"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 17,
   "id": "eaf6fd3c",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/html": [
       "<div>\n",
       "<style scoped>\n",
       "    .dataframe tbody tr th:only-of-type {\n",
       "        vertical-align: middle;\n",
       "    }\n",
       "\n",
       "    .dataframe tbody tr th {\n",
       "        vertical-align: top;\n",
       "    }\n",
       "\n",
       "    .dataframe thead th {\n",
       "        text-align: right;\n",
       "    }\n",
       "</style>\n",
       "<table border=\"1\" class=\"dataframe\">\n",
       "  <thead>\n",
       "    <tr style=\"text-align: right;\">\n",
       "      <th></th>\n",
       "      <th>Date</th>\n",
       "      <th>State</th>\n",
       "      <th>Max_Demand_Met_MW</th>\n",
       "      <th>Shortage_MW</th>\n",
       "      <th>Energy_Met_MU</th>\n",
       "      <th>Drawal_Schedule_MU</th>\n",
       "      <th>OD_UD_MU</th>\n",
       "      <th>Max_OD_MW</th>\n",
       "      <th>Energy_Shortage_MU</th>\n",
       "      <th>Temp_Max</th>\n",
       "      <th>Temp_Min</th>\n",
       "      <th>Temp_Avg</th>\n",
       "      <th>Humidity</th>\n",
       "      <th>Rainfall</th>\n",
       "      <th>Year</th>\n",
       "      <th>Month</th>\n",
       "      <th>Weekday</th>\n",
       "      <th>Season</th>\n",
       "    </tr>\n",
       "  </thead>\n",
       "  <tbody>\n",
       "    <tr>\n",
       "      <th>0</th>\n",
       "      <td>2023-04-01</td>\n",
       "      <td>Andhra Pradesh</td>\n",
       "      <td>10979.0</td>\n",
```

```
"      <td>0.0</td>\n",
"      <td>220.36</td>\n",
"      <td>77.26</td>\n",
"      <td>-0.34</td>\n",
"      <td>383.0</td>\n",
"      <td>0.0</td>\n",
"      <td>33.100000</td>\n",
"      <td>25.200000</td>\n",
"      <td>29.150000</td>\n",
"      <td>77.666667</td>\n",
"      <td>3.933333</td>\n",
"      <td>2023</td>\n",
"      <td>4</td>\n",
"      <td>Saturday</td>\n",
"      <td>Summer</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>1</th>\n",
"      <td>2023-04-02</td>\n",
"      <td>Andhra Pradesh</td>\n",
"      <td>10824.0</td>\n",
"      <td>0.0</td>\n",
"      <td>211.45</td>\n",
"      <td>76.63</td>\n",
"      <td>-1.41</td>\n",
"      <td>382.0</td>\n",
"      <td>0.0</td>\n",
"      <td>33.500000</td>\n",
"      <td>24.800000</td>\n",
"      <td>29.150000</td>\n",
"      <td>77.000000</td>\n",
"      <td>2.900000</td>\n",
"      <td>2023</td>\n",
"      <td>4</td>\n",
"      <td>Sunday</td>\n",
"      <td>Summer</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>2</th>\n",
"      <td>2023-04-03</td>\n",
"      <td>Andhra Pradesh</td>\n",
"      <td>10838.0</td>\n",
"      <td>0.0</td>\n",
"      <td>219.17</td>\n",
"      <td>77.54</td>\n",
"      <td>-0.09</td>\n",
"      <td>548.0</td>\n",
"      <td>0.0</td>\n",
"      <td>34.333333</td>\n",
"      <td>25.166667</td>\n",
"      <td>29.750000</td>\n",
"      <td>76.333333</td>\n",
"      <td>0.833333</td>\n",
"      <td>2023</td>\n",
"      <td>4</td>\n",
"      <td>Monday</td>\n",
"      <td>Summer</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>3</th>\n",
"      <td>2023-04-04</td>\n",
"      <td>Andhra Pradesh</td>\n",
"      <td>11076.0</td>\n",
"      <td>0.0</td>\n",
"      <td>225.07</td>\n",
"      <td>87.67</td>\n",
"      <td>-0.41</td>\n",
"      <td>667.0</td>\n",
"      <td>0.0</td>\n",
"      <td>34.033333</td>\n",
"      <td>25.933333</td>\n",
"      <td>29.983333</td>\n",
```

```
      "      <td>74.666667</td>\n",
      "      <td>2.200000</td>\n",
      "      <td>2023</td>\n",
      "      <td>4</td>\n",
      "      <td>Tuesday</td>\n",
      "      <td>Summer</td>\n",
      "    </tr>\n",
      "    <tr>\n",
      "      <th>4</th>\n",
      "      <td>2023-04-05</td>\n",
      "      <td>Andhra Pradesh</td>\n",
      "      <td>11451.0</td>\n",
      "      <td>0.0</td>\n",
      "      <td>227.39</td>\n",
      "      <td>81.14</td>\n",
      "      <td>0.21</td>\n",
      "      <td>401.0</td>\n",
      "      <td>0.0</td>\n",
      "      <td>35.666667</td>\n",
      "      <td>25.500000</td>\n",
      "      <td>30.583333</td>\n",
      "      <td>73.666667</td>\n",
      "      <td>0.200000</td>\n",
      "      <td>2023</td>\n",
      "      <td>4</td>\n",
      "      <td>Wednesday</td>\n",
      "      <td>Summer</td>\n",
      "    </tr>\n",
      "  </tbody>\n",
      "</table>\n",
      "</div>"
     ],
     "text/plain": [
      "         Date           State  Max_Demand_Met_MW  Shortage_MW  Energy_Met_MU  \\\n",
      "0  2023-04-01  Andhra Pradesh            10979.0          0.0         220.36   \n",
      "1  2023-04-02  Andhra Pradesh            10824.0          0.0         211.45   \n",
      "2  2023-04-03  Andhra Pradesh            10838.0          0.0         219.17   \n",
      "3  2023-04-04  Andhra Pradesh            11076.0          0.0         225.07   \n",
      "4  2023-04-05  Andhra Pradesh            11451.0          0.0         227.39   \n",
      "\n",
      "   Drawal_Schedule_MU  OD_UD_MU  Max_OD_MW  Energy_Shortage_MU   Temp_Max  \\\n",
      "0               77.26     -0.34      383.0                 0.0  33.100000   \n",
      "1               76.63     -1.41      382.0                 0.0  33.500000   \n",
      "2               77.54     -0.09      548.0                 0.0  34.333333   \n",
      "3               87.67     -0.41      667.0                 0.0  34.033333   \n",
      "4               81.14      0.21      401.0                 0.0  35.666667   \n",
      "\n",
      "    Temp_Min   Temp_Avg   Humidity  Rainfall  Year  Month    Weekday  Season  \n",
      "0  25.200000  29.150000  77.666667  3.933333  2023      4   Saturday  Summer  \n",
      "1  24.800000  29.150000  77.000000  2.900000  2023      4     Sunday  Summer  \n",
      "2  25.166667  29.750000  76.333333  0.833333  2023      4     Monday  Summer  \n",
      "3  25.933333  29.983333  74.666667  2.200000  2023      4    Tuesday  Summer  \n",
      "4  25.500000  30.583333  73.666667  0.200000  2023      4  Wednesday  Summer  "
     ]
    },
    "execution_count": 17,
    "metadata": {},
    "output_type": "execute_result"
   }
  ],
  "source": [
   "import pandas as pd\n",
   "df = pd.read_csv(\"PSP_Weather_Merged_EDA_Cleaned.csv\")\n",
   "df.head()"
  ]
 },
 {
  "cell_type": "markdown",
  "id": "a66879c7",
  "metadata": {},
  "source": [
   "## Preprocessing"
```

```
 ]
},
{
 "cell_type": "code",
 "execution_count": 18,
 "id": "f1c0a715",
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Top features by absolute correlation with Max_Demand_Met_MW : ['Temp_Max', 'Temp_Avg', 'Temp_Min'
    "Saved combined scatter/regression figure to top_features_scatter_regression.png\n",
    "Saved combined scatter/regression figure to top_features_scatter_regression.png\n"
   ]
  },
  {
   "data": {
    "image/png": "iVBORw0KGgoAAAANSUhEUgAABKUAAASmCAYAAAD/KRjlAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZl
    "text/plain": [
     "<Figure size 1200x1200 with 6 Axes>"
    ]
   },
   "metadata": {},
   "output_type": "display_data"
  },
  {
   "data": {
    "image/png": "iVBORw0KGgoAAAANSUhEUgAAAk4AAAGACAYAAACNCg6xAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZl
    "text/plain": [
     "<Figure size 600x400 with 1 Axes>"
    ]
   },
   "metadata": {},
   "output_type": "display_data"
  },
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Saved scatter_reg_Temp_Max.png\n"
   ]
  },
  {
   "data": {
    "image/png": "iVBORw0KGgoAAAANSUhEUgAAAk4AAAGACAYAAACNCg6xAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZl
    "text/plain": [
     "<Figure size 600x400 with 1 Axes>"
    ]
   },
   "metadata": {},
   "output_type": "display_data"
  },
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Saved scatter_reg_Temp_Avg.png\n"
   ]
  },
  {
   "data": {
    "image/png": "iVBORw0KGgoAAAANSUhEUgAAAk4AAAGACAYAAACNCg6xAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZl
    "text/plain": [
     "<Figure size 600x400 with 1 Axes>"
    ]
   },
   "metadata": {},
   "output_type": "display_data"
  },
  {
```

```
 "name": "stdout",
 "output_type": "stream",
 "text": [
  "Saved scatter_reg_Temp_Min.png\n"
 ]
},
{
 "data": {
  "image/png": "iVBORw0KGgoAAAANSUhEUgAAAk0AAAGGCAYAAABmPbWyAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZl
  "text/plain": [
   "<Figure size 600x400 with 1 Axes>"
  ]
 },
 "metadata": {},
 "output_type": "display_data"
},
{
 "name": "stdout",
 "output_type": "stream",
 "text": [
  "Saved scatter_reg_Humidity.png\n"
 ]
},
{
 "data": {
  "image/png": "iVBORw0KGgoAAAANSUhEUgAAAk4AAAGGCAYAAACNCg6xAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZl
  "text/plain": [
   "<Figure size 600x400 with 1 Axes>"
  ]
 },
 "metadata": {},
 "output_type": "display_data"
},
{
 "name": "stdout",
 "output_type": "stream",
 "text": [
  "Saved scatter_reg_Rainfall.png\n"
 ]
}
],
"source": [
 "# Scatter + regression plots for top features vs demand\n",
 "import math\n",
 "import pandas as pd\n",
 "import numpy as np\n",
 "import matplotlib.pyplot as plt\n",
 "import seaborn as sns\n",
 "from scipy import stats\n",
 "\n",
 "# Load data\n",
 "df = pd.read_csv('PSP_Weather_Merged_EDA_Cleaned.csv').reset_index(drop=True)\n",
 "if 'Date' in df.columns:\n",
 "    df['Date'] = pd.to_datetime(df['Date'])\n",
 "    # recreate cyclical features if present in preprocessing\n",
 "    df['month'] = df['Date'].dt.month\n",
 "    df['dayofweek'] = df['Date'].dt.dayofweek\n",
 "    df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)\n",
 "    df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)\n",
 "    df['dow_sin'] = np.sin(2 * np.pi * df['dayofweek'] / 7)\n",
 "    df['dow_cos'] = np.cos(2 * np.pi * df['dayofweek'] / 7)\n",
 "\n",
 "TARGET = 'Max_Demand_Met_MW'\n",
 "if TARGET not in df.columns:\n",
 "    raise RuntimeError(f\"Target column '{TARGET}' not found in CSV.\")\n",
 "\n",
 "# Identify candidate feature columns (weather-related heuristics)\n",
 "patterns = ['temp','temperature','humidity','wind','pressure','rain','precip','solar','irradiance']
 "candidate_feats = [c for c in df.columns if any(p in c.lower() for p in patterns) and pd.api.types.
 "# fallback: numeric columns excluding target, Date, State\n",
 "if not candidate_feats:\n",
 "    candidate_feats = [c for c in df.select_dtypes(include=[np.number]).columns if c not in [TARGET
```

```
"    candidate_feats = [c for c in candidate_feats if c.lower() not in ('index','unnamed: 0')]\n",
"\n",
"# Compute global Pearson correlation (abs) with target and pick top features\n",
"corrs = {}\n",
"for f in candidate_feats:\n",
"    try:\n",
"        if df[f].nunique() <= 1:\n",
"            corrs[f] = 0.0\n",
"        else:\n",
"            cor = df[f].corr(df[TARGET])\n",
"            corrs[f] = 0.0 if pd.isna(cor) else float(cor)\n",
"    except Exception:\n",
"        corrs[f] = 0.0\n",
"\n",
"# sort by absolute correlation magnitude\n",
"sorted_feats = sorted(corrs.items(), key=lambda kv: abs(kv[1]), reverse=True)\n",
"top_n = min(6, len(sorted_feats))\n",
"top_features = [f for f,_ in sorted_feats[:top_n]]\n",
"print('Top features by absolute correlation with', TARGET, ':', top_features)\n",
"\n",
"if not top_features:\n",
"    print('No features available for plotting.')\n",
"else:\n",
"    # Create subplots (2 columns)\n",
"    ncols = 2\n",
"    nrows = math.ceil(len(top_features)/ncols)\n",
"    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(12, 4*nrows))\n",
"    axes_flat = np.array(axes).reshape(-1) if hasattr(axes, 'reshape') else np.array([axes]).reshap
"    out_dir_combined = 'top_features_scatter_regression.png'\n",
"    last_i = -1\n",
"    for i, feat in enumerate(top_features):\n",
"        last_i = i\n",
"        ax = axes_flat[i]\n",
"        sub = df[[feat, TARGET]].dropna()\n",
"        # skip if insufficient variation\n",
"        if sub.shape[0] < 3 or sub[feat].nunique() <= 1:\n",
"            ax.text(0.5, 0.5, f'Insufficient data for {feat}', ha='center')\n",
"            ax.set_axis_off()\n",
"            continue\n",
"        sns.regplot(x=feat, y=TARGET, data=sub, scatter_kws={'s':10,'alpha':0.5}, line_kws={'color'
"        # compute Pearson r and annotate\n",
"        try:\n",
"            r, p = stats.pearsonr(sub[feat].astype(float), sub[TARGET].astype(float))\n",
"            ax.annotate(f'r={r:.2f}, p={p:.2g}', xy=(0.02, 0.95), xycoords='axes fraction', va='top
"        except Exception:\n",
"            pass\n",
"        ax.set_title(f'{feat} vs {TARGET}')\n",
"    # hide any unused subplots\n",
"    total_axes = len(axes_flat)\n",
"    for j in range(last_i+1, total_axes):\n",
"        try:\n",
"            axes_flat[j].set_visible(False)\n",
"        except Exception:\n",
"            pass\n",
"    plt.tight_layout()\n",
"    fig.savefig(out_dir_combined, dpi=150, bbox_inches='tight')\n",
"    print(f'Saved combined scatter/regression figure to {out_dir_combined}')\n",
"    plt.show()\n",
"    # Also save individual plots for each top feature\n",
"    for feat in top_features:\n",
"        sub = df[[feat, TARGET]].dropna()\n",
"        if sub.shape[0] < 3 or sub[feat].nunique() <= 1:\n",
"            print(f'Skipping individual plot for {feat} (insufficient data)')\n",
"            continue\n",
"        plt.figure(figsize=(6,4))\n",
"        ax = sns.regplot(x=feat, y=TARGET, data=sub, scatter_kws={'s':15,'alpha':0.6}, line_kws={'c
"        try:\n",
"            r, p = stats.pearsonr(sub[feat].astype(float), sub[TARGET].astype(float))\n",
"            ax.annotate(f'r={r:.2f}, p={p:.2g}', xy=(0.02, 0.95), xycoords='axes fraction', va='top
"        except Exception:\n",
"            pass\n",
"        plt.title(f'{feat} vs {TARGET}')\n",
```

```
  "        out_single = f'scatter_reg_{feat}.png'\n",
  "        plt.tight_layout()\n",
  "        plt.savefig(out_single, dpi=150, bbox_inches='tight')\n",
  "        plt.show()\n",
  "        print(f'Saved {out_single}')\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": 19,
 "id": "281fd789",
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Per-fold results:\n"
   ]
  },
  {
   "data": {
    "text/html": [
     "<div>\n",
     "<style scoped>\n",
     "    .dataframe tbody tr th:only-of-type {\n",
     "        vertical-align: middle;\n",
     "    }\n",
     "\n",
     "    .dataframe tbody tr th {\n",
     "        vertical-align: top;\n",
     "    }\n",
     "\n",
     "    .dataframe thead th {\n",
     "        text-align: right;\n",
     "    }\n",
     "</style>\n",
     "<table border=\"1\" class=\"dataframe\">\n",
     "  <thead>\n",
     "    <tr style=\"text-align: right;\">\n",
     "      <th></th>\n",
     "      <th>fold</th>\n",
     "      <th>model</th>\n",
     "      <th>R2</th>\n",
     "      <th>RMSE</th>\n",
     "      <th>MAE</th>\n",
     "    </tr>\n",
     "  </thead>\n",
     "  <tbody>\n",
     "    <tr>\n",
     "      <th>0</th>\n",
     "      <td>0</td>\n",
     "      <td>LinearRegression</td>\n",
     "      <td>0.991565</td>\n",
     "      <td>590.765979</td>\n",
     "      <td>436.760211</td>\n",
     "    </tr>\n",
     "    <tr>\n",
     "      <th>1</th>\n",
     "      <td>1</td>\n",
     "      <td>LinearRegression</td>\n",
     "      <td>0.987921</td>\n",
     "      <td>640.518199</td>\n",
     "      <td>374.208365</td>\n",
     "    </tr>\n",
     "    <tr>\n",
     "      <th>2</th>\n",
     "      <td>2</td>\n",
     "      <td>LinearRegression</td>\n",
     "      <td>0.994499</td>\n",
     "      <td>714.880030</td>\n",
     "      <td>481.992132</td>\n",
```

```
"    </tr>\n",
"    <tr>\n",
"      <th>3</th>\n",
"      <td>3</td>\n",
"      <td>LinearRegression</td>\n",
"      <td>0.990301</td>\n",
"      <td>259.996294</td>\n",
"      <td>197.709379</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>4</th>\n",
"      <td>4</td>\n",
"      <td>LinearRegression</td>\n",
"      <td>0.991897</td>\n",
"      <td>640.868051</td>\n",
"      <td>431.070005</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>5</th>\n",
"      <td>5</td>\n",
"      <td>LinearRegression</td>\n",
"      <td>0.983112</td>\n",
"      <td>1095.552617</td>\n",
"      <td>656.564168</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>6</th>\n",
"      <td>0</td>\n",
"      <td>RandomForest</td>\n",
"      <td>0.789183</td>\n",
"      <td>2953.358281</td>\n",
"      <td>1740.797193</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>7</th>\n",
"      <td>1</td>\n",
"      <td>RandomForest</td>\n",
"      <td>0.989110</td>\n",
"      <td>608.189909</td>\n",
"      <td>413.846459</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>8</th>\n",
"      <td>2</td>\n",
"      <td>RandomForest</td>\n",
"      <td>0.990221</td>\n",
"      <td>953.146764</td>\n",
"      <td>555.558271</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>9</th>\n",
"      <td>3</td>\n",
"      <td>RandomForest</td>\n",
"      <td>0.994380</td>\n",
"      <td>197.918970</td>\n",
"      <td>108.714664</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>10</th>\n",
"      <td>4</td>\n",
"      <td>RandomForest</td>\n",
"      <td>0.993235</td>\n",
"      <td>585.577971</td>\n",
"      <td>362.632142</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>11</th>\n",
"      <td>5</td>\n",
"      <td>RandomForest</td>\n",
"      <td>0.986730</td>\n",
"      <td>971.115426</td>\n",
"      <td>535.399636</td>\n",
```

```
"    </tr>\n",
"    <tr>\n",
"      <th>12</th>\n",
"      <td>0</td>\n",
"      <td>SVR</td>\n",
"      <td>-0.270043</td>\n",
"      <td>7248.904300</td>\n",
"      <td>4403.487177</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>13</th>\n",
"      <td>1</td>\n",
"      <td>SVR</td>\n",
"      <td>-0.298400</td>\n",
"      <td>6640.924119</td>\n",
"      <td>3977.513919</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>14</th>\n",
"      <td>2</td>\n",
"      <td>SVR</td>\n",
"      <td>-0.512383</td>\n",
"      <td>11853.146042</td>\n",
"      <td>8596.260782</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>15</th>\n",
"      <td>3</td>\n",
"      <td>SVR</td>\n",
"      <td>-0.066003</td>\n",
"      <td>2725.749728</td>\n",
"      <td>2618.185977</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>16</th>\n",
"      <td>4</td>\n",
"      <td>SVR</td>\n",
"      <td>-0.475386</td>\n",
"      <td>8647.672063</td>\n",
"      <td>7303.893804</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>17</th>\n",
"      <td>5</td>\n",
"      <td>SVR</td>\n",
"      <td>-0.174504</td>\n",
"      <td>9136.201454</td>\n",
"      <td>6745.736515</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>18</th>\n",
"      <td>0</td>\n",
"      <td>XGBoost</td>\n",
"      <td>0.763567</td>\n",
"      <td>3127.641496</td>\n",
"      <td>1853.004190</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>19</th>\n",
"      <td>1</td>\n",
"      <td>XGBoost</td>\n",
"      <td>0.979357</td>\n",
"      <td>837.349787</td>\n",
"      <td>574.636356</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>20</th>\n",
"      <td>2</td>\n",
"      <td>XGBoost</td>\n",
"      <td>0.980395</td>\n",
"      <td>1349.533836</td>\n",
"      <td>697.665117</td>\n",
```

```
    "    </tr>\n",
    "    <tr>\n",
    "      <th>21</th>\n",
    "      <td>3</td>\n",
    "      <td>XGBoost</td>\n",
    "      <td>0.993371</td>\n",
    "      <td>214.940782</td>\n",
    "      <td>117.827044</td>\n",
    "    </tr>\n",
    "    <tr>\n",
    "      <th>22</th>\n",
    "      <td>4</td>\n",
    "      <td>XGBoost</td>\n",
    "      <td>0.992992</td>\n",
    "      <td>596.003543</td>\n",
    "      <td>382.114126</td>\n",
    "    </tr>\n",
    "    <tr>\n",
    "      <th>23</th>\n",
    "      <td>5</td>\n",
    "      <td>XGBoost</td>\n",
    "      <td>0.986220</td>\n",
    "      <td>989.609344</td>\n",
    "      <td>520.565380</td>\n",
    "    </tr>\n",
    "  </tbody>\n",
    "</table>\n",
    "</div>"
   ],
   "text/plain": [
    "    fold             model        R2          RMSE          MAE\n",
    "0      0  LinearRegression  0.991565    590.765979   436.760211\n",
    "1      1  LinearRegression  0.987921    640.518199   374.208365\n",
    "2      2  LinearRegression  0.994499    714.880030   481.992132\n",
    "3      3  LinearRegression  0.990301    259.996294   197.709379\n",
    "4      4  LinearRegression  0.991897    640.868051   431.070005\n",
    "5      5  LinearRegression  0.983112   1095.552617   656.564168\n",
    "6      0      RandomForest  0.789183   2953.358281  1740.797193\n",
    "7      1      RandomForest  0.989110    608.189909   413.846459\n",
    "8      2      RandomForest  0.990221    953.146764   555.558271\n",
    "9      3      RandomForest  0.994380    197.918970   108.714664\n",
    "10     4      RandomForest  0.993235    585.577971   362.632142\n",
    "11     5      RandomForest  0.986730    971.115426   535.399636\n",
    "12     0               SVR -0.270043   7248.904300  4403.487177\n",
    "13     1               SVR -0.298400   6640.924119  3977.513919\n",
    "14     2               SVR -0.512383  11853.146042  8596.260782\n",
    "15     3               SVR -0.066003   2725.749728  2618.185977\n",
    "16     4               SVR -0.475386   8647.672063  7303.893804\n",
    "17     5               SVR -0.174504   9136.201454  6745.736515\n",
    "18     0           XGBoost  0.763567   3127.641496  1853.004190\n",
    "19     1           XGBoost  0.979357    837.349787   574.636356\n",
    "20     2           XGBoost  0.980395   1349.533836   697.665117\n",
    "21     3           XGBoost  0.993371    214.940782   117.827044\n",
    "22     4           XGBoost  0.992992    596.003543   382.114126\n",
    "23     5           XGBoost  0.986220    989.609344   520.565380"
   ]
  },
  "metadata": {},
  "output_type": "display_data"
 },
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "Mean cross-fold metrics (lower RMSE better):\n"
  ]
 },
 {
  "data": {
   "text/html": [
    "<div>\n",
    "<style scoped>\n",
```

```
   "    .dataframe tbody tr th:only-of-type {\n",
   "        vertical-align: middle;\n",
   "    }\n",
   "\n",
   "    .dataframe tbody tr th {\n",
   "        vertical-align: top;\n",
   "    }\n",
   "\n",
   "    .dataframe thead th {\n",
   "        text-align: right;\n",
   "    }\n",
   "</style>\n",
   "<table border=\"1\" class=\"dataframe\">\n",
   "  <thead>\n",
   "    <tr style=\"text-align: right;\">\n",
   "      <th></th>\n",
   "      <th>model</th>\n",
   "      <th>R2</th>\n",
   "      <th>RMSE</th>\n",
   "      <th>MAE</th>\n",
   "    </tr>\n",
   "  </thead>\n",
   "  <tbody>\n",
   "    <tr>\n",
   "      <th>0</th>\n",
   "      <td>LinearRegression</td>\n",
   "      <td>0.989882</td>\n",
   "      <td>657.096862</td>\n",
   "      <td>429.717377</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>1</th>\n",
   "      <td>RandomForest</td>\n",
   "      <td>0.957143</td>\n",
   "      <td>1044.884554</td>\n",
   "      <td>619.491394</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>3</th>\n",
   "      <td>XGBoost</td>\n",
   "      <td>0.949317</td>\n",
   "      <td>1185.846465</td>\n",
   "      <td>690.968702</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>2</th>\n",
   "      <td>SVR</td>\n",
   "      <td>-0.299453</td>\n",
   "      <td>7708.766284</td>\n",
   "      <td>5607.513029</td>\n",
   "    </tr>\n",
   "  </tbody>\n",
   "</table>\n",
   "</div>"
  ],
  "text/plain": [
   "              model        R2         RMSE          MAE\n",
   "0  LinearRegression  0.989882   657.096862   429.717377\n",
   "1      RandomForest  0.957143  1044.884554   619.491394\n",
   "3           XGBoost  0.949317  1185.846465   690.968702\n",
   "2               SVR -0.299453  7708.766284  5607.513029"
  ]
 },
 "metadata": {},
 "output_type": "display_data"
},
{
 "name": "stdout",
 "output_type": "stream",
 "text": [
  "Saved summary to time_series_cv_results.csv\n"
 ]
```

```
 },
 {
  "data": {
   "image/png": "iVBORw0KGgoAAAANSUhEUgAAAxYAAAGGCAYAAADmRxfNAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZl
   "text/plain": [
    "<Figure size 800x400 with 1 Axes>"
   ]
  },
  "metadata": {},
  "output_type": "display_data"
 },
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "Best model by mean RMSE: LinearRegression\n"
  ]
 }
],
"source": [
 "# TimeSeriesSplit-based evaluation across folds, summary metrics and simple plots\n",
 "import numpy as np\n",
 "import pandas as pd\n",
 "import matplotlib.pyplot as plt\n",
 "import seaborn as sns\n",
 "from sklearn.model_selection import TimeSeriesSplit, GridSearchCV\n",
 "from sklearn.base import clone\n",
 "from sklearn.linear_model import LinearRegression\n",
 "from sklearn.ensemble import RandomForestRegressor\n",
 "from sklearn.svm import SVR\n",
 "from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error\n",
 "try:\n",
 "    from xgboost import XGBRegressor\n",
 "    has_xgb = True\n",
 "except Exception:\n",
 "    has_xgb = False\n",
 "\n",
 "# Load processed features if not present in memory\n",
 "if 'X_processed' in globals():\n",
 "    X_arr = np.asarray(X_processed)\n",
 "else:\n",
 "    X_arr = np.load('X_processed.npy', allow_pickle=True)\n",
 "\n",
 "if 'y' in globals() and y is not None:\n",
 "    y_arr = np.asarray(y)\n",
 "else:\n",
 "    y_arr = np.asarray(pd.read_csv('PSP_Weather_Merged_EDA_Cleaned.csv')['Max_Demand_Met_MW'])\n",
 "\n",
 "assert X_arr.shape[0] == len(y_arr), 'Feature/target length mismatch'\n",
 "\n",
 "tscv = TimeSeriesSplit(n_splits=6)\n",
 "models = {\n",
 "    'LinearRegression': LinearRegression(),\n",
 "    'RandomForest': RandomForestRegressor(random_state=42),\n",
 "    'SVR': SVR()\n",
 "}\n",
 "if has_xgb:\n",
 "    models['XGBoost'] = XGBRegressor(random_state=42, verbosity=0)\n",
 "\n",
 "# Optional small grid for RandomForest (toggle by setting DO_TUNE=True)\n",
 "DO_TUNE = False\n",
 "rf_grid = {'n_estimators': [50,100], 'max_depth': [None, 10, 20]}\n",
 "\n",
 "fold_results = []\n",
 "fitted_models = {name: [] for name in models.keys()}\n",
 "for fold, (train_idx, test_idx) in enumerate(tscv.split(X_arr)):\n",
 "    X_train, X_test = X_arr[train_idx], X_arr[test_idx]\n",
 "    y_train, y_test = y_arr[train_idx], y_arr[test_idx]\n",
 "    for name, model in models.items():\n",
 "        m = model\n",
 "        try:\n",
 "            if name == 'RandomForest' and DO_TUNE:\n",
```

```
  "                g = GridSearchCV(RandomForestRegressor(random_state=42), rf_grid, cv=3, scoring='ne
  "                g.fit(X_train, y_train)\n",
  "                m = g.best_estimator_\n",
  "            else:\n",
  "                m = clone(model) if 'clone' in globals() else model.__class__(**getattr(model,'get_
  "                m.fit(X_train, y_train)\n",
  "            preds = m.predict(X_test)\n",
  "            r2 = float(r2_score(y_test, preds))\n",
  "            rmse = float(np.sqrt(mean_squared_error(y_test, preds)))\n",
  "            mae = float(mean_absolute_error(y_test, preds))\n",
  "            fold_results.append({'fold': fold, 'model': name, 'R2': r2, 'RMSE': rmse, 'MAE': mae})\
  "            fitted_models[name].append(m)\n",
  "        except Exception as ex:\n",
  "            fold_results.append({'fold': fold, 'model': name, 'error': str(ex)})\n",
  "\n",
  "results_df = pd.DataFrame(fold_results)\n",
  "print('Per-fold results:')\n",
  "display(results_df.sort_values(['model','fold']).reset_index(drop=True))\n",
  "\n",
  "# Aggregate metrics by model\n",
  "agg = results_df.groupby('model').agg({'R2':'mean', 'RMSE':'mean', 'MAE':'mean'}).reset_index()\n",
  "agg = agg.sort_values('RMSE')\n",
  "print('Mean cross-fold metrics (lower RMSE better):')\n",
  "display(agg)\n",
  "agg.to_csv('time_series_cv_results.csv', index=False)\n",
  "print('Saved summary to time_series_cv_results.csv')\n",
  "\n",
  "# Simple visualization: RMSE per model\n",
  "plt.figure(figsize=(8,4))\n",
  "sns.barplot(data=agg, x='model', y='RMSE')\n",
  "plt.title('Mean RMSE by model (TimeSeriesSplit)')\n",
  "plt.tight_layout()\n",
  "plt.savefig('cv_rmse_by_model.png')\n",
  "plt.show()\n",
  "\n",
  "# Choose best model by RMSE (lowest)\n",
  "best_name = agg.iloc[0]['model']\n",
  "print('Best model by mean RMSE:', best_name)\n",
  "# store results and fitted models dict for later cells\n",
  "globals()['cv_results_df'] = results_df\n",
  "globals()['cv_summary'] = agg\n",
  "globals()['fitted_models_cv'] = fitted_models\n",
  "globals()['best_model_name_cv'] = best_name\n",
  "\n",
  "# Note: fitted_models_cv stores models trained on each fold; we'll retrain the chosen estimator on
 ]
},
{
 "cell_type": "markdown",
 "id": "7b26f84c",
 "metadata": {},
 "source": [
  "## Feature/Target Selection"
 ]
},
{
 "cell_type": "code",
 "execution_count": 20,
 "id": "82133413",
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Shapes: X -> (31177, 66)  y -> (31177,)\n"
   ]
  }
 ],
 "source": [
  "# Feature/target selection — use preprocessed data exposed by the preprocessing cell\n",
  "# X (raw features) and y are available from the preprocessing step as globals: X, y\n",
```

```
  "if 'X_processed_df' in globals():\n",
  "    print('Using X_processed_df (DataFrame)')\n",
  "    X_df = X_processed_df\n",
  "    X_arr = X_processed\n",
  "elif 'X_processed' in globals():\n",
  "    X_arr = X_processed\n",
  "    X_df = None\n",
  "else:\n",
  "    X_arr = np.load('X_processed.npy', allow_pickle=True)\n",
  "    X_df = None\n",
  "\n",
  "if 'y' in globals() and y is not None:\n",
  "    y_arr = y\n",
  "else:\n",
  "    y_arr = pd.read_csv('PSP_Weather_Merged_EDA_Cleaned.csv')['Max_Demand_Met_MW']\n",
  "\n",
  "print('Shapes: X ->', getattr(X_arr, 'shape', None), ' y ->', getattr(y_arr, 'shape', None))\n",
  "globals()['X_arr'] = X_arr\n",
  "globals()['X_df'] = X_df\n",
  "globals()['y_arr'] = np.asarray(y_arr)\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": 21,
 "id": "73d397a0",
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Shapes: X_train (24941, 66) X_test (6236, 66) y_train (24941,) y_test (6236,)\n"
   ]
  }
 ],
 "source": [
  "# Robust train/test split that prefers available preprocessed variables\n",
  "from sklearn.model_selection import train_test_split\n",
  "import numpy as np\n",
  "import pandas as pd\n",
  "\n",
  "# Determine feature matrix (try several names in order of preference)\n",
  "if 'X_processed' in globals():\n",
  "    X_for_split = globals().get('X_processed')\n",
  "elif 'X_arr' in globals():\n",
  "    X_for_split = globals().get('X_arr')\n",
  "elif 'X_processed_df' in globals():\n",
  "    X_for_split = globals().get('X_processed_df').values\n",
  "else:\n",
  "    try:\n",
  "        X_for_split = np.load('X_processed.npy', allow_pickle=True)\n",
  "    except Exception:\n",
  "        X_for_split = None\n",
  "\n",
  "# Determine target vector\n",
  "if 'y' in globals() and globals().get('y') is not None:\n",
  "    y_for_split = globals().get('y')\n",
  "elif 'y_arr' in globals():\n",
  "    y_for_split = globals().get('y_arr')\n",
  "else:\n",
  "    try:\n",
  "        y_for_split = pd.read_csv('PSP_Weather_Merged_EDA_Cleaned.csv')['Max_Demand_Met_MW'].values\n",
  "    except Exception:\n",
  "        y_for_split = None\n",
  "\n",
  "if X_for_split is None or y_for_split is None:\n",
  "    raise RuntimeError('Could not find suitable X and/or y for train_test_split. Run preprocessing/\n",
  "\n",
  "# Ensure numpy arrays\n",
  "X_for_split = np.asarray(X_for_split)\n",
  "y_for_split = np.asarray(y_for_split)\n",
```

```
  "\n",
  "# Do the split and expose variables to globals for subsequent cells\n",
  "X_train, X_test, y_train, y_test = train_test_split(X_for_split, y_for_split, test_size=0.2, random
  "globals()['X_train'] = X_train\n",
  "globals()['X_test'] = X_test\n",
  "globals()['y_train'] = y_train\n",
  "globals()['y_test'] = y_test\n",
  "print('Shapes: X_train', getattr(X_train, 'shape', None), 'X_test', getattr(X_test, 'shape', None),
 ]
},
{
 "cell_type": "code",
 "execution_count": 22,
 "id": "40c3dc9d",
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "y_train type: <class 'numpy.ndarray'>\n",
    "y_train shape: (24941,)\n",
    "first 10 values: [  151.   162. 13135.  2466. 14700.   236. 18372.   189. 25994.  2466.]\n"
   ]
  }
 ],
 "source": [
  "# Safely inspect y_train whether it's a pandas Series or numpy array\n",
  "import numpy as np\n",
  "from IPython.display import display\n",
  "\n",
  "if 'y_train' not in globals():\n",
  "    print('y_train is not defined — run the train/test split cell first.')\n",
  "else:\n",
  "    yt = globals().get('y_train')\n",
  "    print('y_train type:', type(yt))\n",
  "    try:\n",
  "        if hasattr(yt, 'head'):\n",
  "            display(yt.head())\n",
  "        else:\n",
  "            arr = np.asarray(yt)\n",
  "            print('y_train shape:', getattr(arr, 'shape', None))\n",
  "            print('first 10 values:', arr[:10])\n",
  "    except Exception as e:\n",
  "        print('Could not display y_train:', e)"
 ]
},
{
 "cell_type": "markdown",
 "id": "62fcced9",
 "metadata": {},
 "source": [
  "## Model Training & Evaluation"
 ]
},
{
 "cell_type": "code",
 "execution_count": 23,
 "id": "f6c04623",
 "metadata": {},
 "outputs": [
  {
   "data": {
    "text/plain": [
     "{'Linear Regression': {'R2': 0.99621852415649,\n",
     "  'RMSE': 454.2412435188631,\n",
     "  'MAPE': 0.24138426046319758},\n",
     " 'Random Forest': {'R2': 0.9986159389437038,\n",
     "  'RMSE': 274.81062723377164,\n",
     "  'MAPE': 0.028613318462261828},\n",
     " 'SVR': {'R2': 0.08565557380979161,\n",
     "  'RMSE': 7063.35135039826,\n",
```

```
      "  'MAPE': 5.1656904935757},\n",
      " 'XGBoost': {'R2': 0.9986454144145782,\n",
      "  'RMSE': 271.86864570725044,\n",
      "  'MAPE': 0.03753184761973767}}"
     ]
    },
    "execution_count": 23,
    "metadata": {},
    "output_type": "execute_result"
   }
  ],
  "source": [
   "from sklearn.linear_model import LinearRegression\n",
   "from sklearn.ensemble import RandomForestRegressor\n",
   "from sklearn.svm import SVR\n",
   "try:\n",
   "    from xgboost import XGBRegressor\n",
   "    has_xgb = True\n",
   "except Exception:\n",
   "    has_xgb = False\n",
   "from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_percentage_error\n",
   "import numpy as np\n",
   "\n",
   "models = {\n",
   "    \"Linear Regression\": LinearRegression(),\n",
   "    \"Random Forest\": RandomForestRegressor(),\n",
   "    \"SVR\": SVR()\n",
   "}\n",
   "if has_xgb:\n",
   "    models['XGBoost'] = XGBRegressor()\n",
   "\n",
   "results = {}\n",
   "for name, model in models.items():\n",
   "    model.fit(X_train, y_train)\n",
   "    preds = model.predict(X_test)\n",
   "    results[name] = {\n",
   "        \"R2\": float(r2_score(y_test, preds)),\n",
   "        \"RMSE\": float(np.sqrt(mean_squared_error(y_test, preds))),\n",
   "        \"MAPE\": float(mean_absolute_percentage_error(y_test, preds))\n",
   "    }\n",
   "\n",
   "results"
  ]
 },
 {
  "cell_type": "markdown",
  "id": "cb0f394a",
  "metadata": {},
  "source": [
   "## Best Model Selection"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 24,
  "id": "4b6891e1",
  "metadata": {},
  "outputs": [
   {
    "data": {
     "text/plain": [
      "'XGBoost'"
     ]
    },
    "execution_count": 24,
    "metadata": {},
    "output_type": "execute_result"
   }
  ],
  "source": [
   "best = max(results, key=lambda x: results[x][\"R2\"])\n",
   "best"
```

```
 ]
},
{
 "cell_type": "markdown",
 "id": "6cc4182d",
 "metadata": {},
 "source": [
  "## Train Best Model Fully"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "40f87d86",
 "metadata": {},
 "outputs": [
  {
   "ename": "SyntaxError",
   "evalue": "invalid syntax (2326919692.py, line 51)",
   "output_type": "error",
   "traceback": [
    "\u001b[1;36m  Cell \u001b[1;32mIn[26], line 51\u001b[1;36m\u001b[0m\n\u001b[1;33m    :\u001b[0m\n
   ]
  }
 ],
 "source": [
  "# Train selected best model on the full dataset and save it\n",
  "import numpy as np\n",
  "import pandas as pd\n",
  "import pickle\n",
  "\n",
  "if 'best' not in globals():\n",
  "    raise RuntimeError('Best model name not found — run the best-selection cell first.')\n",
  "if 'models' not in globals():\n",
  "    raise RuntimeError('Models dictionary not found — ensure the model definitions cell has run.')\
  "if best not in models:\n",
  "    raise RuntimeError(f\"Selected best model '{best}' not found in models dict. Available: {list(m
  "est = models[best]\n",
  "# Find X and y (preferred variables or fallbacks)\n",
  "X_full = globals().get('X_processed') if 'X_processed' in globals() else globals().get('X_arr') if
  "if X_full is None:\n",
  "    try:\n",
  "        X_full = np.load('X_processed.npy', allow_pickle=True)\n",
  "    except Exception:\n",
  "        X_full = None\n",
  "y_full = globals().get('y') if 'y' in globals() and globals().get('y') is not None else globals().g
  "if y_full is None:\n",
  "    try:\n",
  "        y_full = pd.read_csv('PSP_Weather_Merged_EDA_Cleaned.csv')['Max_Demand_Met_MW'].values\n",
  "    except Exception:\n",
  "        y_full = None\n",
  "if X_full is None or y_full is None:\n",
  "    raise RuntimeError('Could not find suitable X and/or y to train the full model. Run preprocessi
  "X_full = np.asarray(X_full)\n",
  "y_full = np.asarray(y_full).ravel()\n",
  "if X_full.shape[0] != len(y_full):\n",
  "    raise RuntimeError(f'Feature/target length mismatch: X rows={X_full.shape[0]} vs y length={len(
  "# Fit and persist\n",
  "est.fit(X_full, y_full)\n",
  "globals()['best_model'] = est\n",
  "with open('demand_model.pkl', 'wb') as f:\n",
  "    pickle.dump(est, f)\n",
  "print(f'Trained and saved best model: {best} (n={X_full.shape[0]}) -> demand_model.pkl')"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "d94cdfda",
 "metadata": {},
 "outputs": [
  {
```

```
 "name": "stdout",
 "output_type": "stream",
 "text": [
  "Top 10 states by predicted demand:\n"
 ]
},
{
 "data": {
  "text/html": [
   "<div>\n",
   "<style scoped>\n",
   "    .dataframe tbody tr th:only-of-type {\n",
   "        vertical-align: middle;\n",
   "    }\n",
   "\n",
   "    .dataframe tbody tr th {\n",
   "        vertical-align: top;\n",
   "    }\n",
   "\n",
   "    .dataframe thead th {\n",
   "        text-align: right;\n",
   "    }\n",
   "</style>\n",
   "<table border=\"1\" class=\"dataframe\">\n",
   "  <thead>\n",
   "    <tr style=\"text-align: right;\">\n",
   "      <th></th>\n",
   "      <th>State</th>\n",
   "      <th>predicted_demand</th>\n",
   "    </tr>\n",
   "  </thead>\n",
   "  <tbody>\n",
   "    <tr>\n",
   "      <th>0</th>\n",
   "      <td>Maharashtra</td>\n",
   "      <td>24267.15</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>1</th>\n",
   "      <td>Uttar Pradesh</td>\n",
   "      <td>20541.57</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>2</th>\n",
   "      <td>Gujarat</td>\n",
   "      <td>19655.95</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>3</th>\n",
   "      <td>Tamil Nadu</td>\n",
   "      <td>16804.91</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>4</th>\n",
   "      <td>Karnataka</td>\n",
   "      <td>13418.99</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>5</th>\n",
   "      <td>Rajasthan</td>\n",
   "      <td>11740.83</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>6</th>\n",
   "      <td>Madhya Pradesh</td>\n",
   "      <td>11343.44</td>\n",
   "    </tr>\n",
   "    <tr>\n",
   "      <th>7</th>\n",
   "      <td>Telangana</td>\n",
   "      <td>10961.71</td>\n",
   "    </tr>\n",
```

```
    "    <tr>\n",
    "      <th>8</th>\n",
    "      <td>Andhra Pradesh</td>\n",
    "      <td>10166.58</td>\n",
    "    </tr>\n",
    "    <tr>\n",
    "      <th>9</th>\n",
    "      <td>West Bengal</td>\n",
    "      <td>8708.18</td>\n",
    "    </tr>\n",
    "  </tbody>\n",
    "</table>\n",
    "</div>"
   ],
   "text/plain": [
    "           State  predicted_demand\n",
    "0     Maharashtra          24267.15\n",
    "1   Uttar Pradesh          20541.57\n",
    "2         Gujarat          19655.95\n",
    "3      Tamil Nadu          16804.91\n",
    "4       Karnataka          13418.99\n",
    "5       Rajasthan          11740.83\n",
    "6  Madhya Pradesh          11343.44\n",
    "7       Telangana          10961.71\n",
    "8  Andhra Pradesh          10166.58\n",
    "9     West Bengal           8708.18"
   ]
  },
  "metadata": {},
  "output_type": "display_data"
 },
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "State with highest predicted demand: Maharashtra\n",
   "Saved state-level predictions to state_level_predictions.csv\n"
  ]
 }
],
"source": [
 "# State-level prediction: which State has the highest predicted demand?\n",
 "import pandas as pd\n",
 "import numpy as np\n",
 "import joblib\n",
 "import pickle\n",
 "from IPython.display import display\n",
 "\n",
 "# Load preprocessing pipeline and trained model\n",
 "try:\n",
 "    pipeline = joblib.load('preprocess.joblib')\n",
 "except Exception as e:\n",
 "    raise RuntimeError('Could not load preprocess.joblib. Run the preprocessing cell first.') from
 "\n",
 "try:\n",
 "    model = pickle.load(open('demand_model.pkl', 'rb'))\n",
 "except Exception as e:\n",
 "    raise RuntimeError('Could not load demand_model.pkl. Train and save the model first.') from e\n
 "\n",
 "# Read raw data and create the same engineered date features used in preprocessing\n",
 "df_raw = pd.read_csv('PSP_Weather_Merged_EDA_Cleaned.csv').dropna().reset_index(drop=True)\n",
 "if 'Date' in df_raw.columns:\n",
 "    df_raw['Date'] = pd.to_datetime(df_raw['Date'])\n",
 "    # ensure cyclical features exist\n",
 "    df_raw['month'] = df_raw['Date'].dt.month\n",
 "    df_raw['dayofweek'] = df_raw['Date'].dt.dayofweek\n",
 "    df_raw['month_sin'] = np.sin(2 * np.pi * df_raw['month'] / 12)\n",
 "    df_raw['month_cos'] = np.cos(2 * np.pi * df_raw['month'] / 12)\n",
 "    df_raw['dow_sin'] = np.sin(2 * np.pi * df_raw['dayofweek'] / 7)\n",
 "    df_raw['dow_cos'] = np.cos(2 * np.pi * df_raw['dayofweek'] / 7)\n",
 "\n",
 "# Verify State column exists\n",
```

```
"if 'State' not in df_raw.columns:\n",
"    raise RuntimeError(\"No 'State' column found in the CSV.\")\n",
"\n",
"# Representative row per state: use the last available record (by Date) to reflect recent condition
"if 'Date' in df_raw.columns:\n",
"    rep = df_raw.sort_values('Date').groupby('State', as_index=False).last()\n",
"else:\n",
"    rep = df_raw.groupby('State', as_index=False).last()\n",
"\n",
"# Drop target column if present and Date (pipeline expects same input columns as during fit)\n",
"TARGET = 'Max_Demand_Met_MW'\n",
"if TARGET in rep.columns:\n",
"    rep = rep.drop(columns=[TARGET])\n",
"if 'Date' in rep.columns:\n",
"    rep = rep.drop(columns=['Date'])\n",
"\n",
"# Transform representative rows using the saved pipeline — make this robust to missing columns\n",
"try:\n",
"    X_state = rep.copy()\n",
"    # Attempt to infer which input columns the pipeline expects\n",
"    cat_cols = []\n",
"    num_cols = []\n",
"    try:\n",
"        pre = pipeline.named_steps.get('preprocess', pipeline)\n",
"        # transformers_ is available after fit; each entry is (name, transformer, columns)\n",
"        for tname, trans, cols in getattr(pre, 'transformers_', []):\n",
"            if isinstance(cols, (list, tuple)):\n",
"                if tname == 'cat':\n",
"                    cat_cols = list(cols)\n",
"                elif tname == 'num':\n",
"                    num_cols = list(cols)\n",
"    except Exception:\n",
"        # best-effort fallback — we'll use all columns present in rep\n",
"        cat_cols = [c for c in rep.columns if rep[c].dtype == object]\n",
"        num_cols = [c for c in rep.columns if rep[c].dtype.kind in 'biufc']\n",
"\n",
"    # Ensure the expected columns exist in X_state; fill with defaults when missing\n",
"    for c in cat_cols:\n",
"        if c not in X_state.columns:\n",
"            X_state[c] = 'Unknown'\n",
"    for c in num_cols:\n",
"        if c not in X_state.columns:\n",
"            X_state[c] = 0.0\n",
"\n",
"    # Reorder columns to the order the pipeline saw during fit if available\n",
"    input_cols = None\n",
"    try:\n",
"        input_cols = list(pipeline.feature_names_in_)\n",
"    except Exception:\n",
"        # try to build from cat_cols + num_cols\n",
"        if cat_cols or num_cols:\n",
"            input_cols = cat_cols + num_cols\n",
"    if input_cols is not None:\n",
"        missing = [c for c in input_cols if c not in X_state.columns]\n",
"        for c in missing:\n",
"            # conservative default\n",
"            X_state[c] = 0 if (c not in cat_cols) else 'Unknown'\n",
"        X_state = X_state[input_cols]\n",
"\n",
"    Xp = pipeline.transform(X_state)\n",
"    if hasattr(Xp, 'toarray'):\n",
"        Xp = Xp.toarray()\n",
"except Exception as e:\n",
"    raise RuntimeError('Error transforming state-level features with the saved pipeline: ' + str(e)
"\n",
"# Predict\n",
"try:\n",
"    preds = model.predict(Xp)\n",
"except Exception as e:\n",
"    raise RuntimeError('Error predicting with the loaded model: ' + str(e)) from e\n",
"\n",
"rep['predicted_demand'] = preds\n",
```

```
    "rep_sorted = rep.sort_values('predicted_demand', ascending=False).reset_index(drop=True)\n",
    "\n",
    "print('Top 10 states by predicted demand:')\n",
    "display(rep_sorted[['State','predicted_demand']].head(10))\n",
    "print('State with highest predicted demand:', rep_sorted.iloc[0]['State'])\n",
    "\n",
    "# Save predictions\n",
    "rep_sorted[['State','predicted_demand']].to_csv('state_level_predictions.csv', index=False)\n",
    "print('Saved state-level predictions to state_level_predictions.csv')"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.8"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```