Forecating Energy Consumption - Kansas City, Kansas

Project Summary:

- This project focuses on forecasting energy consumption for Kansas City, Kansas. Data from the Energy Information Administration (EIA) and weather data from Visual Crossing are used. The goal is to build a machine learning model capable of predicting energy use specifically for this region, using historical data from 2020 to 2024.

- The dataset combines hourly energy consumption records and weather data. Key features such as hour, day of the week, month, and specific holidays, are created to capture time-based patterns. Temperature is included to account for weather-related variability.

- An XGBoost regression model is trained and evaluated using time-series cross-validation. On the test set it achieves an average RMSE of approximately 5.84% of the energy consumption range. The model is then used to predict future energy consumption for Kansas City, Kansas. This tool aims to support energy management and utility planning for the region.

In [1]:
```
import holidays

import requests
import json

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
color_pal = sns.color_palette()

from statsmodels.tsa.stattools import adfuller
```

```python
import pmdarima as pm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV
from sklearn.utils import resample
```

API Call for Energy Data (eia.gov)

In [2]:
```python
with open('eia_api_key') as f:
    api_key = f.read()
```

In [3]:
```python
base_url = ('https://api.eia.gov/v2/electricity/rto/region-sub-ba-data/data/'
            '?frequency=hourly&data[0]=value&facets[subba][]=KACY&start=2020-01-01T00'
            '&end=2024-12-28T22&sort[0][column]=period&sort[0][direction]=asc'
            f'&length=5000&api_key={api_key}')
```

In [4]:
```python
offset = 0
all_data = []

while True:
    url = f"{base_url}&offset={offset}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()['response']['data']
        if not data:
            break
        all_data.extend(data)
        offset += 5000
    else:
        print(f"Request failed with status code: {response.status_code}")
        print(response.text)
        break
```

make dataframe

In [5]:
```python
df = pd.DataFrame(all_data)
```

In [6]:
```python
df.head(3)
```

Out[6]:

	period	subba	subba-name	parent	parent-name	value	value-units
0	2020-01-01T00	KACY	Kansas City Board of Public Utilities - SWPP	SWPP	Southwest Power Pool	259	megawatthours
1	2020-01-01T01	KACY	Kansas City Board of Public Utilities - SWPP	SWPP	Southwest Power Pool	263	megawatthours
2	2020-01-01T02	KACY	Kansas City Board of Public Utilities - SWPP	SWPP	Southwest Power Pool	258	megawatthours

In [7]:
```python
df['value'] = df.value.astype(int)
df['period'] = pd.to_datetime(df['period'])
```

In [8]:
```python
df = df[['period','value']]
df.rename(columns={'period':'datetime'}, inplace=True)
df.head()
```

Out[8]:

	datetime	value
0	2020-01-01 00:00:00	259
1	2020-01-01 01:00:00	263
2	2020-01-01 02:00:00	258
3	2020-01-01 03:00:00	253
4	2020-01-01 04:00:00	247

In [9]:
```python
print(f'Missing: {df.isna().sum().sum()}')
print(f'Duplicates: {df.index.duplicated().sum()}')
```

```
Missing: 0
Duplicates: 0
```

Hourly Weather Data (visualcrossing.com)

In [10]:
```python
df2 = pd.read_csv(r'C:\Users\baile\Downloads\kansas city, kansas 2020-01-01 to 2024-12-28.csv')
```

In [11]:
```python
df2.head(3)
```

Out[11]:

	name	datetime	temp	feelslike	dew	humidity	precip	precipprob	preciptype	snow	...	sealevelpressure	cloudcover
0	kansas city, kansas	2020-01-01T00:00:00	-2.1	-6.2	-6.1	74.00	0.0	0	NaN	0.0	...	1014.5	0.0
1	kansas city, kansas	2020-01-01T01:00:00	-0.7	-3.4	-5.7	68.95	0.0	0	NaN	0.0	...	1013.6	0.0
2	kansas city, kansas	2020-01-01T02:00:00	-1.2	-1.2	-6.2	68.87	0.0	0	NaN	0.0	...	1012.5	0.0

3 rows × 24 columns

In [12]:
```python
df2['datetime'] = pd.to_datetime(df2['datetime'])
df2 = df2[['datetime','temp']]
```

In [13]:
```python
df2.head()
```

Out[13]:

	datetime	temp
0	2020-01-01 00:00:00	-2.1
1	2020-01-01 01:00:00	-0.7
2	2020-01-01 02:00:00	-1.2
3	2020-01-01 03:00:00	-0.7
4	2020-01-01 04:00:00	-0.1

In [14]:
```
print(f'Missing: {df2.isna().sum().sum()}')
print(f'Duplicates: {df2.index.duplicated().sum()}')
```

```
Missing: 0
Duplicates: 0
```

Merge Energy and Weather Data on Datetime

In [15]:
```
df = pd.merge(df, df2, on='datetime', how='inner')
```

In [16]:
```
df = df.set_index('datetime')
```

In [17]:
```
df.head()
```

Out[17]:

datetime	value	temp
2020-01-01 00:00:00	259	-2.1
2020-01-01 01:00:00	263	-0.7
2020-01-01 02:00:00	258	-1.2
2020-01-01 03:00:00	253	-0.7
2020-01-01 04:00:00	247	-0.1

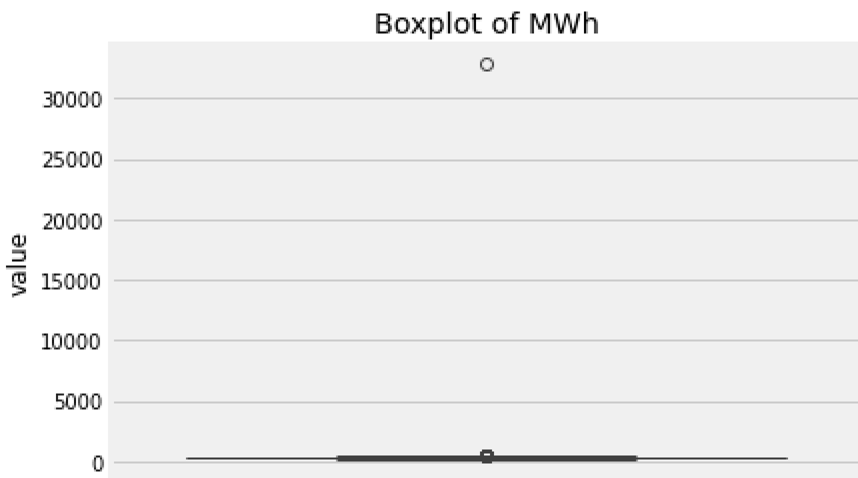
In [18]:
```
print(f'Missing: {df.isna().sum().sum()}')
print(f'Duplicates: {df.index.duplicated().sum()}')
```

```
Missing: 0
Duplicates: 5
```

In [19]:
```
df = df[~df.index.duplicated(keep='first')]
```

Visualizing MWh Usage

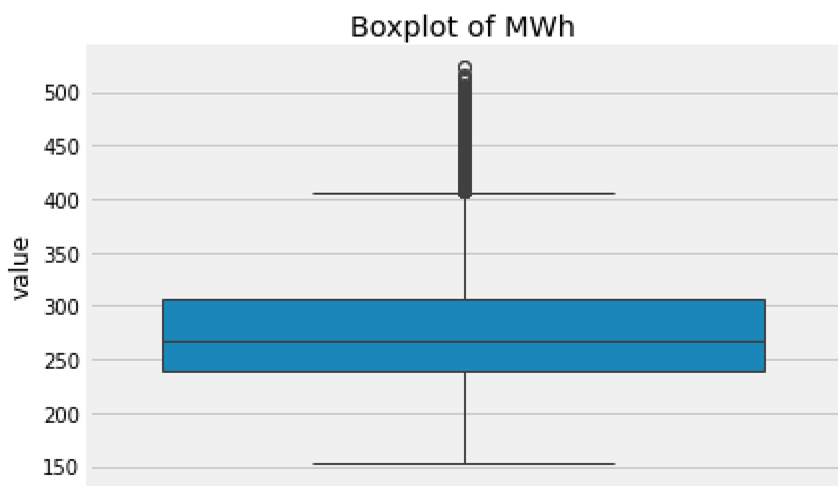
In [20]:
```
sns.boxplot(df.value)
plt.title('Boxplot of MWh');
```



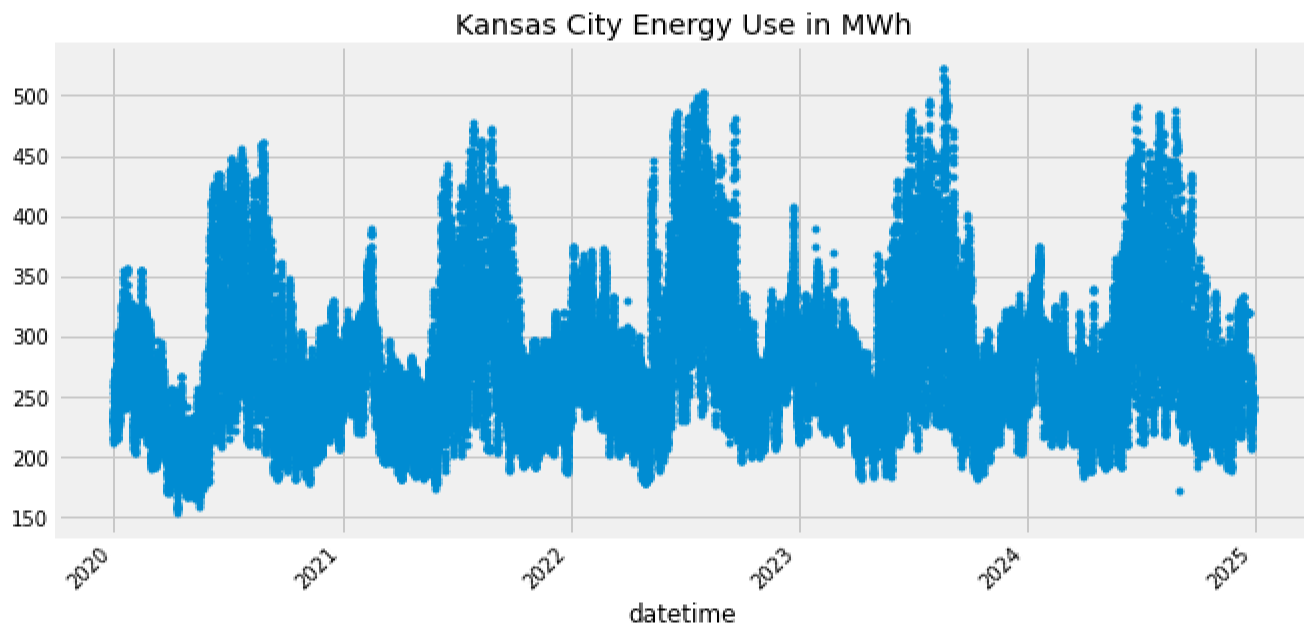
- This value being so extreme makes it seem like an error, so it will be dropped.

In [21]:
```
df = df[df.value<30000]
```

In [22]:
```
sns.boxplot(df.value)
plt.title('Boxplot of MWh');
```



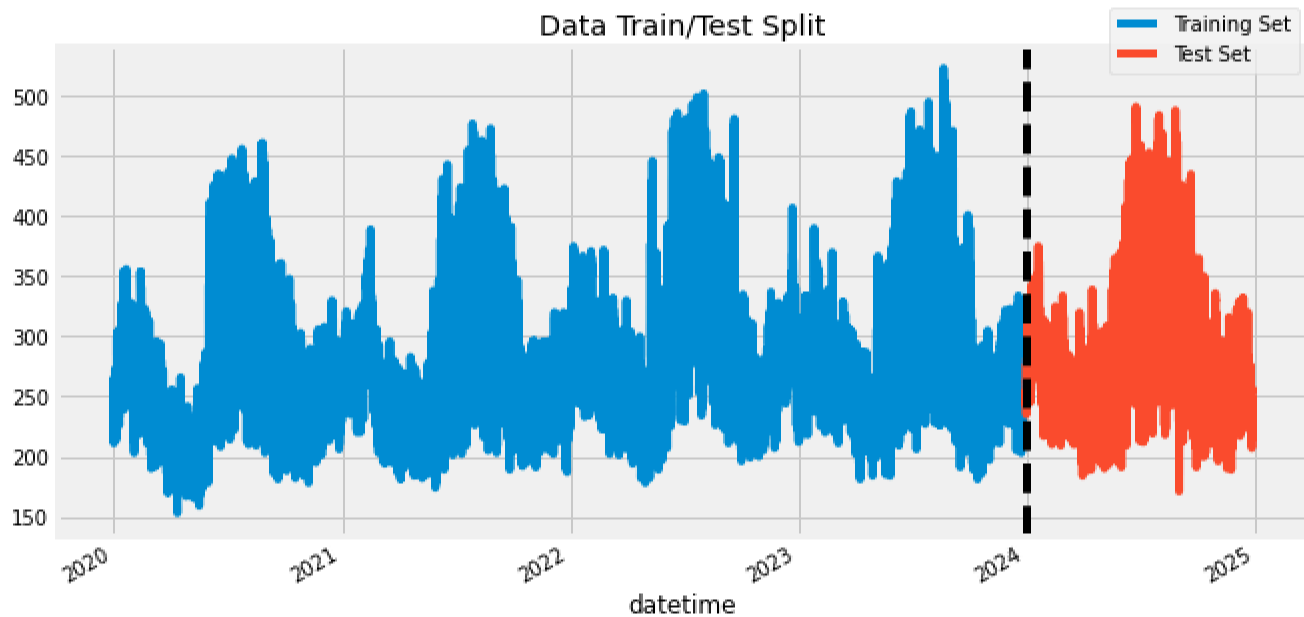

```
In [23]: df.value.plot(style='.', figsize=(10,5), color=color_pal[0], title='Kansas City Energy Use in MWh')
plt.xticks(rotation=45);
```



Train/Test Split

In [24]:
```
train = df.loc[df.index < '01-01-2024']
test = df.loc[df.index >= '01-01-2024']
```

In [25]:
```
fig,ax = plt.subplots(figsize=(10,5))
train.value.plot(ax=ax, label='Training Set', title='Data Train/Test Split')
test.value.plot(ax=ax, label='Test Set')
ax.axvline('01-01-2024',color='black',ls='--')
ax.legend(['Training Set','Test Set'], loc='upper right', bbox_to_anchor=(1, 1.1));
```



- The test set is the last 362 days of the data. Next an ARIMA model will be built as a baseline.

ARIMA

arima dataframe - aggregated biweekly

In [26]:
```
arima_df = df['value'].resample('2W').mean().to_frame()
arima_df = np.log(arima_df)
```

In [27]:
```
train_mask = arima_df.index < '2024-01-01'
test_mask = arima_df.index >= '2024-01-01'

arima_train = arima_df[train_mask]
arima_test = arima_df[test_mask]
```

check for stationarity

In [28]:
```
def adfuller(df):
    adf_test = adfuller(df)
    if adf_test[1] < 0.05:
        print('Data is stationary')
    else:
        print('Data is not stationary')
```

In [ ]:
```
adfuller(arima_train)
```

In [ ]:
```

```

Feature Creation

In [26]:
```
def create_features(df):
    '''create times series features'''
    df['hour'] = df.index.hour
    df['dayofweek'] = df.index.day_of_week
    df['quarter'] = df.index.quarter
    df['month'] = df.index.month
    df['year'] = df.index.year
    df['dayofyear'] = df.index.dayofyear

    us_holidays = holidays.US()
    specific_holidays = {"New Year's Day", "Labor Day", "Thanksgiving", "Christmas Day"}
```

```python
    df['is_specific_holiday'] = df.index.map(lambda x: us_holidays.get(x.date()) in specific_holidays)

    return df

create_features(df);
```

- Creating features such as hour and is_specific_holiday should make for a better model.

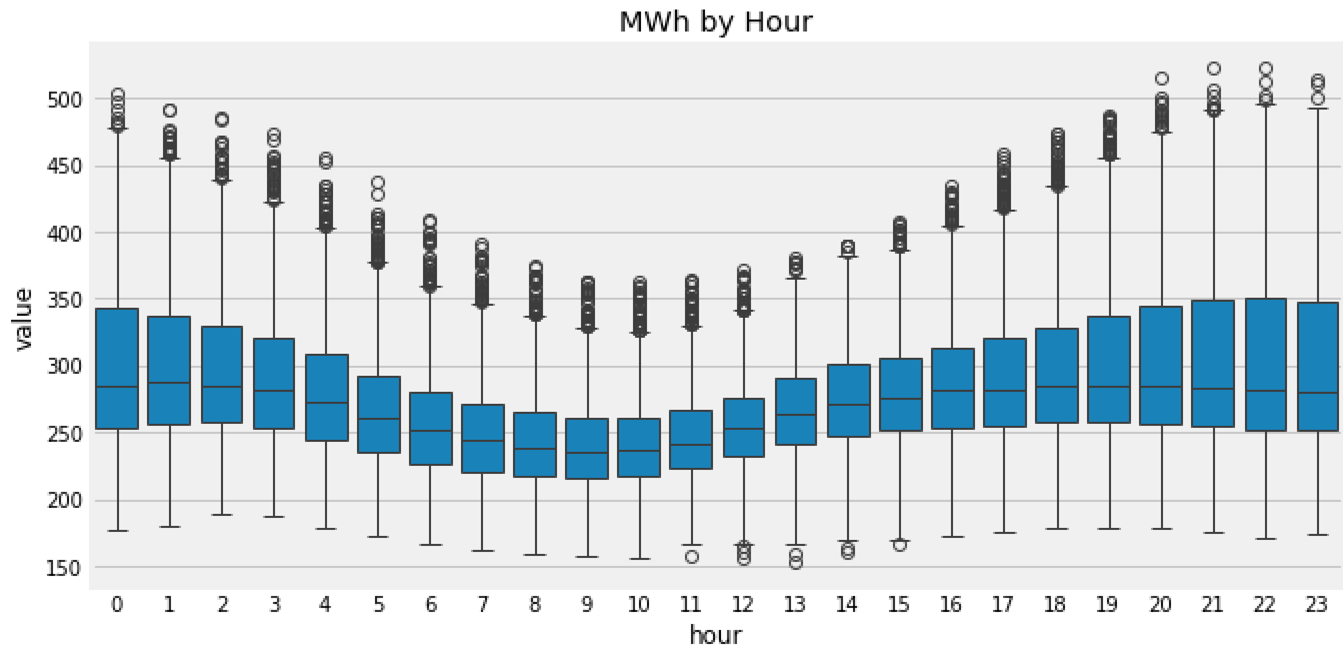
In [27]:
```python
df.head(3)
```

Out[27]:

datetime	value	temp	hour	dayofweek	quarter	month	year	dayofyear	is_specific_holiday
2020-01-01 00:00:00	259	-2.1	0	2	1	1	2020	1	True
2020-01-01 01:00:00	263	-0.7	1	2	1	1	2020	1	True
2020-01-01 02:00:00	258	-1.2	2	2	1	1	2020	1	True

Visualizing Feature/Target Relationships

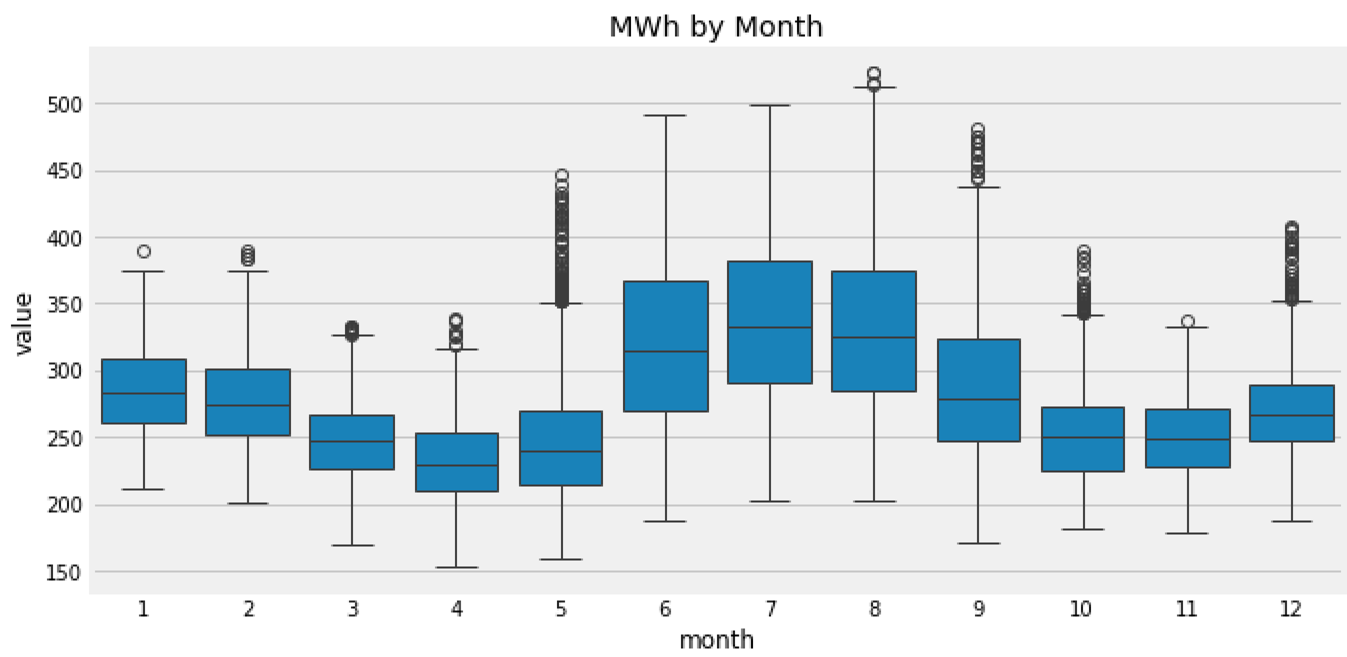
In [28]:
```python
plt.figure(figsize=(10,5))
sns.boxplot(data=df, x='hour',y='value')
plt.title('MWh by Hour');
```



- Energy consumption appears to be lowest from about 8 to 10 in the morning, and highest from about 8 to 11 at night.

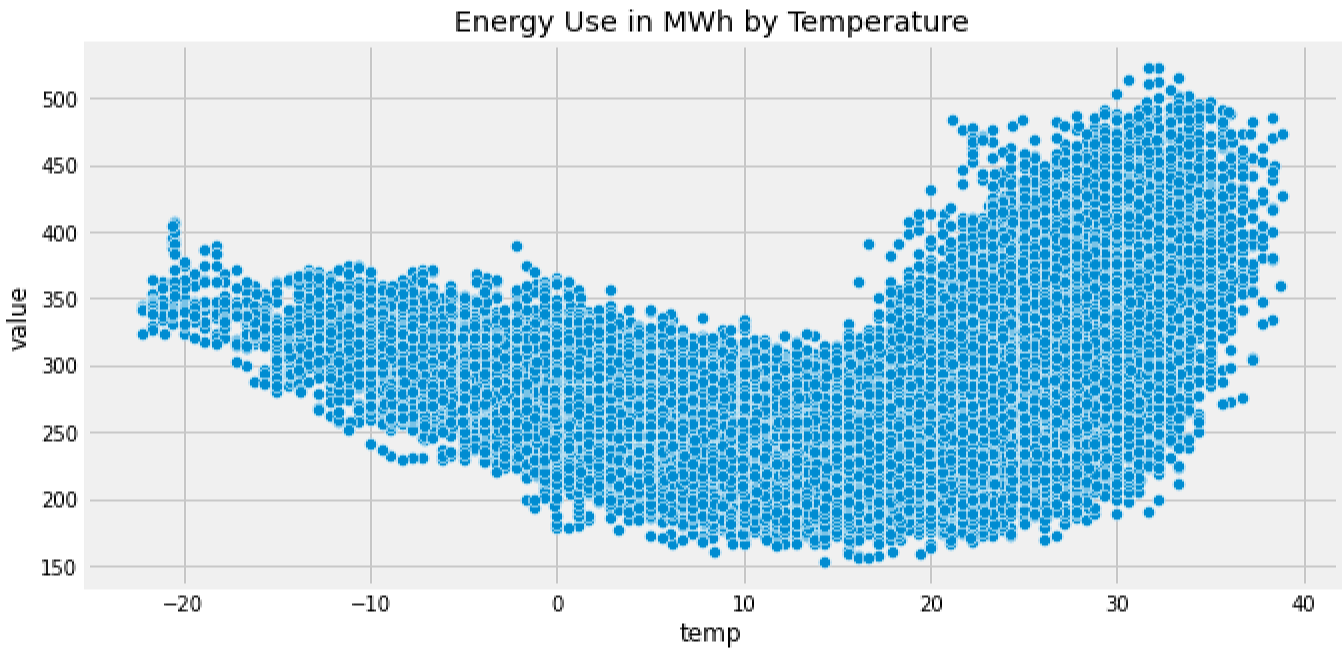
In [29]:

```
plt.figure(figsize=(10,5))
sns.boxplot(data=df, x='month',y='value')
plt.title('MWh by Month');
```



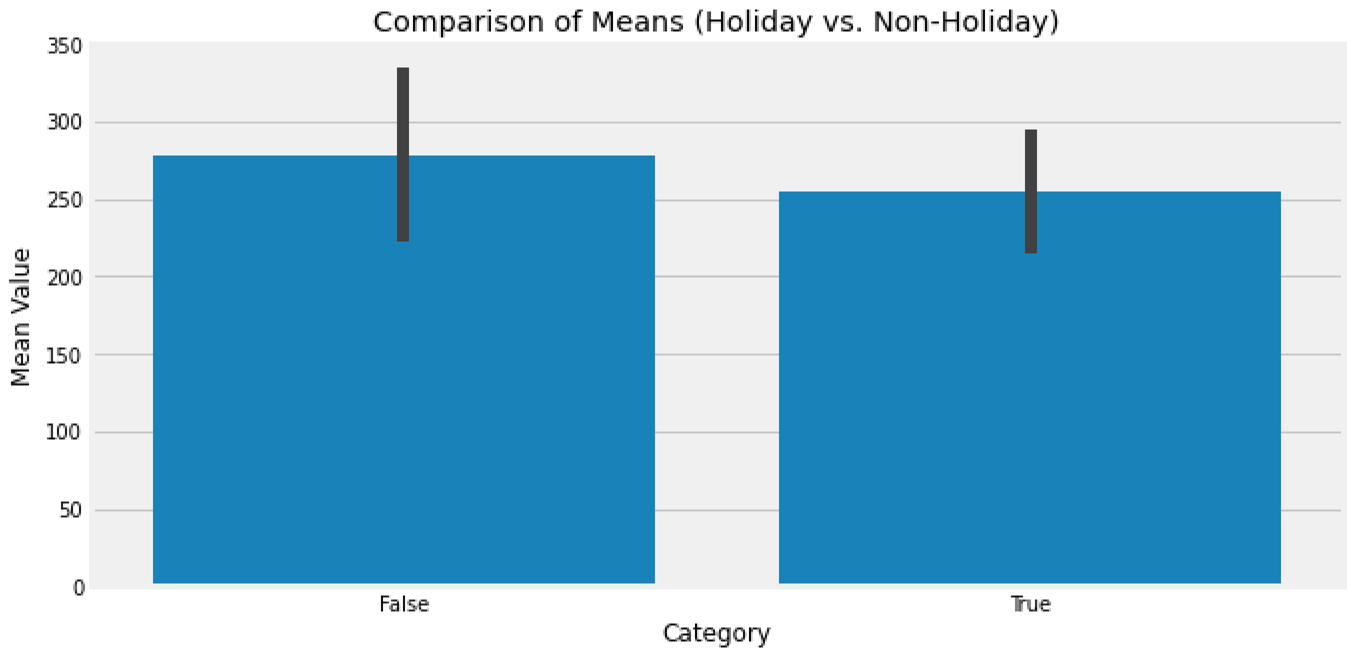
- The peaks in energy consumption during summer and the dips in fall align with changes in temperature.

In [30]:
```
plt.figure(figsize=(10,5))
sns.scatterplot(x=df.temp, y=df.value)
plt.title('Energy Use in MWh by Temperature');
```



- Energy usage appears to be lowest in mild temperatures and highest in hot temperatures. The reason for this graph not being u-shaped could be explained by natural gas being widely used in Kansas City during the winter.

In [31]:
```
plt.figure(figsize=(10,5))
sns.barplot(data=df, x='is_specific_holiday', y='value', errorbar='sd')
plt.xlabel('Category')
plt.ylabel('Mean Value')
plt.title('Comparison of Means (Holiday vs. Non-Holiday)');
```



- Mean energy usage does not appear to be higher on holidays. That's probably because the specified holidays: New Year's Day, Labor Day, Thanksgiving, Christmas Day, are around winter.

XGBoost Model

In [32]:
```
train = create_features(train.copy())
test = create_features(test.copy())

features = ['hour', 'dayofweek', 'quarter', 'month', 'year', 'dayofyear','temp', 'is_specific_holiday']
target = 'value'

X_train = train[features]
y_train = train[target]
```

```
X_test = test[features]
y_test = test[target]
```

In [33]:
```
reg = xgb.XGBRegressor(n_estimators=1000, early_stopping_rounds=50, random_state=1)
reg.fit(X_train,y_train,
        eval_set=[(X_train,y_train), (X_test,y_test)],
        verbose=100)
```

```
[0]	validation_0-rmse:198.72407	validation_1-rmse:202.91748
[67]	validation_0-rmse:12.34317	validation_1-rmse:22.76310
```

Out[33]:

XGBRegressor

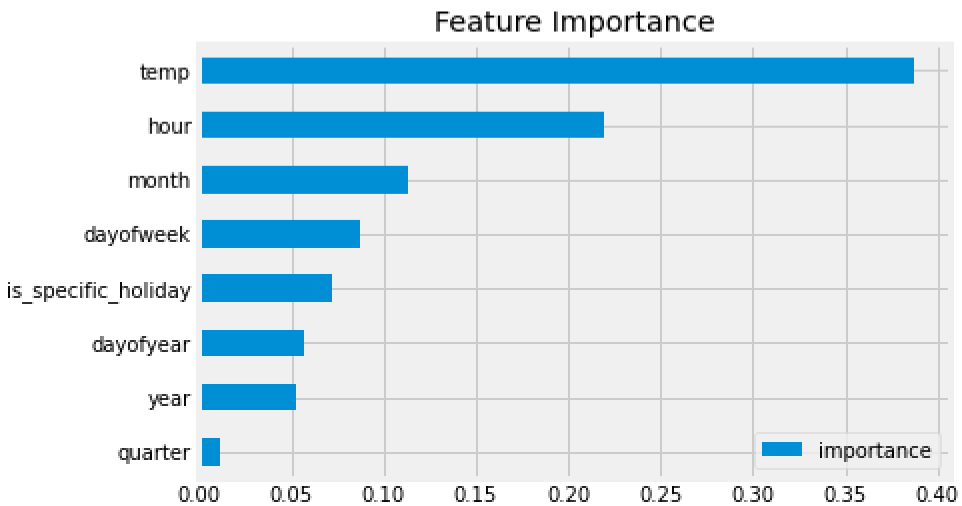
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
```

- Early stopping terminated the training process after 68 iterations. This indicates that the validation set error stopped improving further. Additional iterations would lead to overfitting on the training data.

Feature Importance

In [34]:
```
importance_df = pd.DataFrame(data=reg.feature_importances_,
             index=reg.feature_names_in_,
             columns=['importance'] )
```

In [35]:
```
importance_df.sort_values('importance').plot(kind='barh',title='Feature Importance');
```

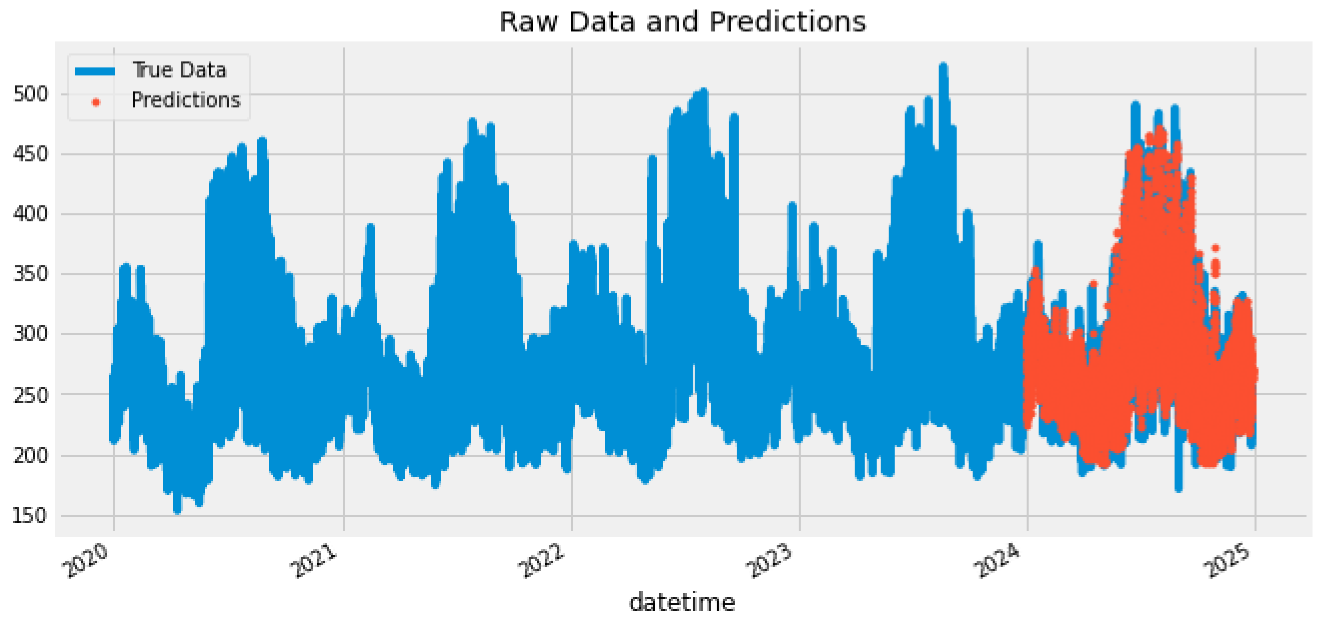


- Unsurprisingly, the temperature data turned out to be the most important feature.

Forecast Made on the Test Set

In [36]:
```
test['prediction'] = reg.predict(X_test)
df = df.merge(test[['prediction']], how='left', left_index=True, right_index=True)
```

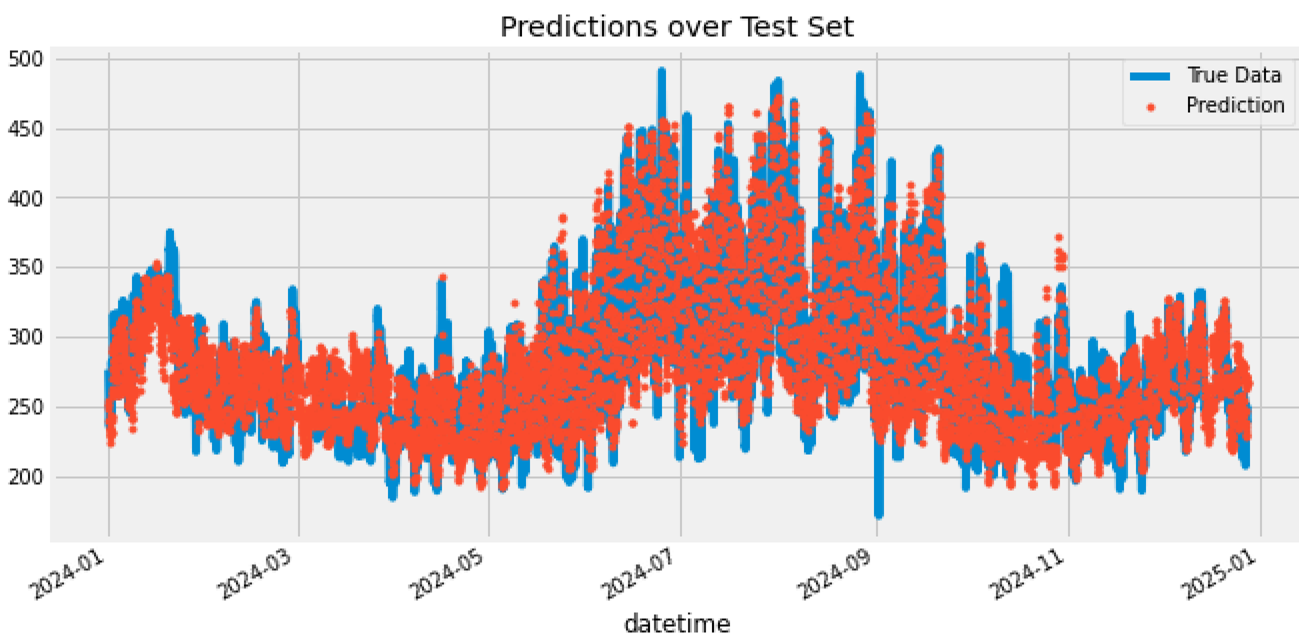
In [37]:
```
ax = df[['value']].plot(figsize=(10,5))
df['prediction'].plot(ax=ax, style='.')
plt.legend(['True Data','Predictions'])
ax.set_title('Raw Data and Predictions');
```



In [38]:
```
ax = df.loc[(df.index>'01-01-2024') & (df.index<'12-28-2024')]['value']\
.plot(figsize=(10,5), title='Predictions over Test Set')

df.loc[(df.index>'01-01-2024') & (df.index<'12-28-2024')]['prediction'].plot(style='.')

plt.legend(['True Data','Prediction']);
```



- Visually, the fit looks pretty good. A quantitative measure like rmse would be more informative though.

Time Series Cross Val & Test Score

In [39]:
```
X = df.drop(columns=['value','prediction'])
y = df.value
```

In [40]:
```
tscv = TimeSeriesSplit(n_splits=5)

rmse_scores = []

for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

```
    reg.fit(
        X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],verbose=0)

    y_pred = reg.predict(X_test)

    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    rmse_scores.append(rmse)

rmse_scores_rounded = [round(score, 2) for score in rmse_scores]
print("RMSE scores for each fold:", rmse_scores_rounded)
print("Average RMSE:", np.mean(rmse_scores))
```

```
RMSE scores for each fold: [24.93, 26.8, 22.73, 24.25, 21.6]
Average RMSE: 24.061008290912728
```

- The cross val scores are close enough together to indicate that the model's predictions are stable across different subsets of the data.

In [56]:
```
final_model = xgb.XGBRegressor(n_estimators=1000, early_stopping_rounds=50, random_state=1)
final_model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose=0)

y_pred_test = final_model.predict(X_test)
test_rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))
print(f'The test RMSE is {test_rmse.round(2)}.')
```

```
The test RMSE is 21.6.
```

In [42]:
```
my_range = df.value.max()-df.value.min()
avg_error_perc = round((test_rmse/my_range)*100,2)

print(f"The model's performance on the test set indicates an average error of {avg_error_perc}%.")
```

```
The model's performance on the test set indicates an average error of 5.84%.
```

Future Dataframe

- The future dataframe will be temeprature predictions in Kansas City for the next 16 days along with the created features used earlier.

In [43]:
```
future_df = pd.read_csv(r'C:\Users\baile\Downloads\kansas city, kansas 2024-12-28 to 2025-01-12.csv')
```

In [44]:
```
future_df = future_df[['datetime','temp']]
future_df['datetime'] = pd.to_datetime(future_df['datetime'])
future_df = future_df.set_index('datetime')
future_df.head()
```

Out[44]:

	temp
datetime	
2024-12-28 00:00:00	5.0
2024-12-28 01:00:00	4.4
2024-12-28 02:00:00	6.1
2024-12-28 03:00:00	7.2
2024-12-28 04:00:00	7.2

In [45]:
```
future_df = create_features(future_df)
```

In [46]:
```
future_df.head()
```

Out[46]:

datetime	temp	hour	dayofweek	quarter	month	year	dayofyear	is_specific_holiday
2024-12-28 00:00:00	5.0	0	5	4	12	2024	363	False
2024-12-28 01:00:00	4.4	1	5	4	12	2024	363	False
2024-12-28 02:00:00	6.1	2	5	4	12	2024	363	False
2024-12-28 03:00:00	7.2	3	5	4	12	2024	363	False
2024-12-28 04:00:00	7.2	4	5	4	12	2024	363	False

Predicting Future Energy Consumption(2024-12-28 to 2025-01-12)

bootstrap resampling for confidence intervals

In [47]:

```
future_predictions = []
for _ in range(100):
    X_boot, y_boot = resample(X_train, y_train)

    reg = xgb.XGBRegressor(n_estimators=68)
    reg.fit(X_boot, y_boot, verbose=0)

    pred = reg.predict(future_df)
    future_predictions.append(pred)
```

In [48]:

```
future_predictions = np.array(future_predictions)
lower_bound = np.percentile(future_predictions, 2.5, axis=0)
upper_bound = np.percentile(future_predictions, 97.5, axis=0)
```

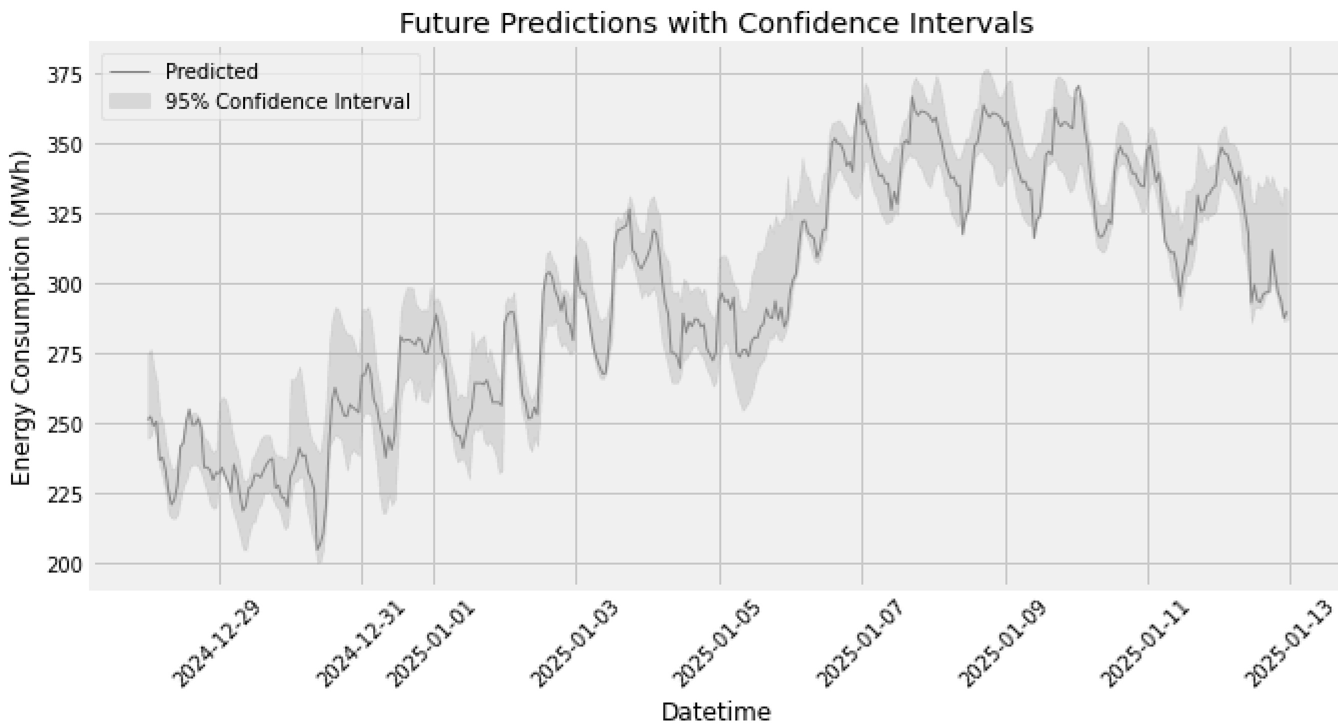
plot

In [49]:
```
future_df['pred'] = reg.predict(future_df)

plt.figure(figsize=(10, 5))

plt.plot(future_df.index, future_df['pred'], color=color_pal[4], ms=1, lw=1, label='Predicted')
plt.fill_between(future_df.index,lower_bound,upper_bound,color='gray',alpha=0.2,label='95% Confidence Interval')

plt.title('Future Predictions with Confidence Intervals')
plt.xlabel('Datetime')
plt.ylabel('Energy Consumption (MWh)')
plt.xticks(rotation=45)
plt.legend();
```



- The plot above shows the predicted energey use in MWh in Kansas City from 2024-12-28 to 2025-01-12.

---

predictions dataframe

In [50]:
```
mean_predictions = future_predictions.mean(axis=0)

predictions_df = pd.DataFrame({
    'datetime': future_df.index,
    'pred': mean_predictions,
    'lower_bound': lower_bound,
    'upper_bound': upper_bound  })

predictions_df.set_index('datetime', inplace=True)
```

In [51]:
```
predictions_df.head()
```

Out[51]:

datetime	pred	lower_bound	upper_bound
2024-12-28 00:00:00	261.541901	244.706550	274.953561
2024-12-28 01:00:00	262.395569	245.559415	276.466607
2024-12-28 02:00:00	259.488403	248.880390	269.882551
2024-12-28 03:00:00	255.511734	245.261096	265.157507
2024-12-28 04:00:00	244.360703	234.923753	253.269280

predictions_df.to_csv('kc_energy_preds_2024_12_28_to_2025_01_12.csv')