```
import streamlit as st
import pandas as pd
import pickle
import numpy as np
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import smtplib
from email.message import EmailMessage
from io import StringIO
import time
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image

# ----------------- CONFIG -----------------
st.set_page_config(
    page_title="⚡ Smart Load Forecaster",
    layout="wide",
    page_icon="⚡",
    initial_sidebar_state="expanded"
)

# ----------------- Custom CSS -----------------
st.markdown("""
<style>
    :root {
        --primary: #4a6bff;
        --secondary: #ff6b6b;
        --accent: #6bff6b;
```

```
    --dark: #2E3A59;
    --light: #F9F9F9;
    --gradient-start: #6a11cb;
    --gradient-end: #2575fc;
}

.main {
    background-color: var(--light);
    color: var(--dark);
}

.header {
    font-size: 2.5rem;
    color: var(--primary);
    font-weight: 800;
    margin-bottom: 0.5rem;
    background: linear-gradient(90deg, var(--gradient-start), var(--gradient-end));
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
}

.subheader {
    font-size: 1.5rem;
    color: var(--primary);
    font-weight: 600;
    margin-bottom: 1rem;
}

.metric-card {
    background: white;
    border-radius: 12px;
```

```
    padding: 20px;
    box-shadow: 0 6px 12px rgba(0,0,0,0.08);
    margin-bottom: 20px;
    border-left: 4px solid var(--primary);
    transition: transform 0.3s;
}

.metric-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 16px rgba(0,0,0,0.12);
}

.metric-title {
    font-size: 1rem;
    color: var(--dark);
    font-weight: 600;
    margin-bottom: 8px;
}

.metric-value {
    font-size: 2rem;
    color: var(--primary);
    font-weight: 700;
}

.positive {
    color: #28a745;
}

.negative {
    color: #dc3545;
```

```
}

.stButton>button {
    background: linear-gradient(90deg, var(--gradient-start), var(--gradient-end));
    color: white;
    border-radius: 12px;
    padding: 10px 20px;
    font-weight: 600;
    transition: all 0.3s;
    border: none;
    box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}

.stButton>button:hover {
    transform: translateY(-2px);
    box-shadow: 0 6px 12px rgba(0,0,0,0.15);
    background: linear-gradient(90deg, var(--gradient-start), var(--gradient-end));
}

.stDownloadButton>button {
    background: linear-gradient(90deg, #00b09b, #96c93d) !important;
}

.error-box {
    background-color: #ffebee;
    border-left: 4px solid #f44336;
    padding: 1rem;
    margin: 1rem 0;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
}
```

```
.success-box {
    background-color: #e8f5e9;
    border-left: 4px solid #4caf50;
    padding: 1rem;
    margin: 1rem 0;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
}

.info-box {
    background-color: #e3f2fd;
    border-left: 4px solid #2196f3;
    padding: 1rem;
    margin: 1rem 0;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
}

.stDataFrame {
    border-radius: 12px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.08);
}

.stSelectbox div[data-baseweb="select"] {
    border-radius: 12px !important;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
}

.stFileUploader>div {
    border: 2px dashed var(--primary) !important;
```

```
    border-radius: 12px !important;
    background-color: rgba(74, 107, 255, 0.05);
    transition: all 0.3s;
}

.stFileUploader>div:hover {
    background-color: rgba(74, 107, 255, 0.1);
    border-color: var(--gradient-end) !important;
}

.feature-card {
    background: white;
    border-radius: 12px;
    padding: 15px;
    margin-bottom: 15px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.08);
    border-left: 4px solid var(--accent);
}

.feature-title {
    font-weight: 600;
    color: var(--dark);
    margin-bottom: 5px;
}

.feature-value {
    font-size: 1.2rem;
    color: var(--primary);
    font-weight: 700;
}
```

```
.tab-container {
    background: white;
    border-radius: 12px;
    padding: 20px;
    box-shadow: 0 4px 12px rgba(0,0,0,0.08);
    margin-top: 20px;
}

/* Custom scrollbar */
::-webkit-scrollbar {
    width: 8px;
}

::-webkit-scrollbar-track {
    background: #f1f1f1;
    border-radius: 10px;
}

::-webkit-scrollbar-thumb {
    background: var(--primary);
    border-radius: 10px;
}

::-webkit-scrollbar-thumb:hover {
    background: #3a56ff;
}

/* Custom tabs */
.stTabs [data-baseweb="tab-list"] {
    gap: 10px;
}
```

```
.stTabs [data-baseweb="tab"] {
    background: white;
    border-radius: 8px 8px 0 0 !important;
    padding: 10px 20px !important;
    transition: all 0.3s;
    border: 1px solid #e0e0e0 !important;
    margin-right: 0 !important;
}

.stTabs [data-baseweb="tab"]:hover {
    background: #f5f7ff !important;
    color: var(--primary) !important;
}

.stTabs [aria-selected="true"] {
    background: var(--primary) !important;
    color: white !important;
    border-color: var(--primary) !important;
}

/* Custom number input */
.stNumberInput>div>div>input {
    border-radius: 12px !important;
}

/* Custom slider */
.stSlider>div>div>div>div {
    background: var(--primary) !important;
}
```

```
    /* Custom checkbox */
    .stCheckbox>label {
        font-weight: 500;
    }

    /* Custom text input */
    .stTextInput>div>div>input {
        border-radius: 12px !important;
    }
</style>
""", unsafe_allow_html=True)

# ----------------- Load the trained models -----------------
@st.cache_data
def load_model(model_type="XGBoost"):
    try:
        if model_type == "XGBoost":
            with open(r"C:\Users\lenovo\load fore\xgb_model (1).pkl", "rb") as f:
                return pickle.load(f)
        elif model_type == "Random Forest":
            with open(r"C:\Users\lenovo\load fore\rf_model.pkl", "rb") as f:
                return pickle.load(f)
        elif model_type == "LSTM":
            with open(r"C:\Users\lenovo\load fore\lstm_model.pkl", "rb") as f:
                return pickle.load(f)
    except Exception as e:
        st.error(f"❌ Error loading model: {str(e)}")
        return None

# ----------------- Helper Functions -----------------
def create_cyclical_features(df):
```

```
    """Create cyclical time features"""
    if 'hour' in df.columns:
        df['hour_sin'] = np.sin(2 * np.pi * df['hour']/24)
        df['hour_cos'] = np.cos(2 * np.pi * df['hour']/24)
    if 'dayofweek' in df.columns:
        df['dayofweek_sin'] = np.sin(2 * np.pi * df['dayofweek']/7)
        df['dayofweek_cos'] = np.cos(2 * np.pi * df['dayofweek']/7)
    return df

def calculate_error_metrics(y_true, y_pred):
    """Calculate all error metrics"""
    metrics = {
        'R² Score': r2_score(y_true, y_pred),
        'MAE': mean_absolute_error(y_true, y_pred),
        'RMSE': np.sqrt(mean_squared_error(y_true, y_pred)),
        'MAPE': np.mean(np.abs((y_true - y_pred) / (y_true + 1e-10))) * 100,
        'Max Error': np.max(np.abs(y_true - y_pred))
    }
    return metrics

def send_email(receiver_email, df, report_title, include_metrics=True, include_charts=True):
    """Send forecast results via email"""
    try:
        msg = EmailMessage()
        msg['Subject'] = f"⚡ Load Forecasting Report: {report_title}"
        msg['From'] = "your-email@gmail.com"  # Replace with your email
        msg['To'] = receiver_email

        # Create HTML content
        html = f"""
        <html>
```

```
        <body>
            <h2 style="color: #4a6bff;">Load Forecasting Report: {report_title}</h2>
            <p>Attached is your load forecasting report generated on {datetime.now().strftime('%Y-%m-%d %H:%M')}</p>
            <p>Summary Statistics:</p>
            <ul>
                <li>Records: {len(df)}</li>
                <li>Time Period: {df['datetime'].min()} to {df['datetime'].max()}</li>
                <li>Average Predicted Load: {df['Predicted Load'].mean():.2f}</li>
            </ul>
            <p>Best regards,<br>Smart Load Forecaster Team</p>
        </body>
    </html>
    """
    msg.add_alternative(html, subtype='html')

    # Attach CSV
    buffer = StringIO()
    df.to_csv(buffer, index=False)
    msg.add_attachment(buffer.getvalue(), filename="forecast_report.csv", subtype="csv")

    # Send email
    with smtplib.SMTP_SSL('smtp.gmail.com', 465) as smtp:
        smtp.login("your-email@gmail.com", "your-app-password")  # Replace with credentials
        smtp.send_message(msg)
    return True
  except Exception as e:
    st.error(f"Email error: {str(e)}")
    return False

def plot_error_distribution(df):
```

```
"""Create error distribution plots"""
fig = make_subplots(rows=1, cols=2, subplot_titles=("Error Distribution", "Error Over Time"))

# Histogram
fig.add_trace(
    go.Histogram(
        x=df['Error'],
        nbinsx=50,
        marker_color='#4a6bff',
        opacity=0.7,
        name='Errors'
    ),
    row=1, col=1
)

# Error over time
fig.add_trace(
    go.Scatter(
        x=df['datetime'],
        y=df['Error'],
        mode='markers',
        marker=dict(
            color=df['Error'],
            colorscale='RdYlGn',
            showscale=True,
            size=8,
            opacity=0.7
        ),
        name='Errors'
    ),
    row=1, col=2
```

```
    )

    fig.update_layout(
        title_text="Prediction Error Analysis",
        showlegend=False,
        height=500,
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)'
    )

    # Update xaxis properties
    fig.update_xaxes(title_text="Error Value", row=1, col=1)
    fig.update_xaxes(title_text="Datetime", row=1, col=2)

    # Update yaxis properties
    fig.update_yaxes(title_text="Count", row=1, col=1)
    fig.update_yaxes(title_text="Error Value", row=1, col=2)

    return fig

def plot_feature_importance(model, features):
    """Plot feature importance if available"""
    try:
        if hasattr(model, 'feature_importances_'):
            importance = model.feature_importances_
            indices = np.argsort(importance)[::-1]

            fig = go.Figure()
            fig.add_trace(go.Bar(
                x=[features[i] for i in indices],
                y=importance[indices],
```

```
            marker_color='#4a6bff',
            opacity=0.8
        ))

        fig.update_layout(
            title="Feature Importance",
            xaxis_title="Features",
            yaxis_title="Importance",
            height=500,
            plot_bgcolor='rgba(0,0,0,0)',
            paper_bgcolor='rgba(0,0,0,0)'
        )

        return fig
    except:
        return None

# ---------- UI Elements ----------
# Header with gradient
st.markdown("""
<div class="header">⚡ Smart Load Forecaster</div>
<p style="font-size: 1.1rem; color: #5D6D7E; margin-bottom: 2rem;">
AI-Powered Electricity Load Forecasting with Advanced Analytics
</p>
""", unsafe_allow_html=True)

# Sidebar for settings
with st.sidebar:
    # Logo and title
    st.markdown("""
    <div style="text-align: center; margin-bottom: 2rem;">
```

```
    <h2 style="color: var(--primary); margin-bottom: 0.5rem;">⚡</h2>
    <h3 style="color: var(--dark); margin-top: 0;">Smart Forecaster</h3>
</div>
""", unsafe_allow_html=True)

# Model selection
st.markdown("### 🔧 Model Settings")
selected_model = st.selectbox(
    "Select Forecasting Model",
    ["XGBoost", "Random Forest", "LSTM"],
    help="Choose the machine learning model for predictions"
)

forecast_horizon = st.slider(
    "Forecast Horizon (hours)",
    1, 168, 24,
    help="Number of hours to forecast into the future"
)

confidence_interval = st.slider(
    "Confidence Interval (%)",
    80, 99, 95,
    help="Confidence level for prediction intervals"
)

st.markdown("---")
st.markdown("### 📊 Visualization Settings")
theme = st.selectbox(
    "Chart Theme",
    ["Plotly", "Seaborn", "GGPlot", "Dark"],
    index=0
```

```
    )

    st.markdown("---")
    st.markdown("### ℹ️ About")
    st.markdown("""
    This app uses advanced machine learning to predict electricity load based on:
    - Historical consumption patterns
    - Temporal features (hour, day, month)
    - Rolling statistics
    - Weather data (if available)
    """)

    st.markdown("---")
    st.markdown("""
    <div style="font-size: 0.8rem; color: #777;">
    Developed by Energy Analytics Team<br>
    Version 2.0.0
    </div>
    """, unsafe_allow_html=True)

# ---------- File Upload ----------
upload_col1, upload_col2 = st.columns([3, 1])
with upload_col1:
    uploaded_file = st.file_uploader(
        "📤 Upload your CSV file with historical load data",
        type=["csv"],
        help="File should include datetime and consumption columns"
    )

with upload_col2:
    st.markdown("""
```

```
    <div style="text-align: center; margin-top: 1.5rem;">
        <a href="https://example.com/sample_data.csv" download style="text-decoration: none;">
            <button style="background: linear-gradient(90deg, #00b09b, #96c93d); color: white; border: none; padding: 10px 15px; border-radius: 12px; font-weight: 600; cursor: pointer; transition: all 0.3s;">
                Download Sample Data
            </button>
        </a>
    </div>
    """, unsafe_allow_html=True)


if uploaded_file is not None:
    try:
        df = pd.read_csv(uploaded_file)

        # Convert datetime if present
        if 'datetime' in df.columns:
            df['datetime'] = pd.to_datetime(df['datetime'])
            df = df.sort_values('datetime')

        # Create time-based features
        if 'datetime' in df.columns:
            df['hour'] = df['datetime'].dt.hour
            df['dayofweek'] = df['datetime'].dt.dayofweek
            df['month'] = df['datetime'].dt.month
            df['is_weekend'] = df['dayofweek'].isin([5,6]).astype(int)

        # Feature engineering
        df = create_cyclical_features(df)

        # Required columns
        required_columns = ['hour', 'dayofweek', 'month', 'is_weekend',
```

```
            'lag_1', 'lag_2', 'lag_3', 'lag_24', 'lag_48', 'lag_72',
            'rolling_mean_3', 'rolling_mean_24', 'rolling_std_24',
            'hour_sin', 'hour_cos']

# Check for missing columns
missing = [col for col in required_columns if col not in df.columns]
if missing:
    st.error(f"⚠ Missing required columns: {', '.join(missing)}")
    st.stop()

# ---------- Model Prediction ----------
model = load_model(selected_model)
if model is None:
    st.error("Failed to load the selected model")
    st.stop()

with st.spinner(f"🔮 Generating predictions using {selected_model}..."):
    time.sleep(1)  # Simulate processing

    # Make predictions
    X = df[required_columns]
    predictions = model.predict(X)
    df['Predicted Load'] = predictions

    # Calculate metrics if actual values available
    if 'load' in df.columns:
        metrics = calculate_error_metrics(df['load'], df['Predicted Load'])

    # ---------- Results Display ----------
    st.success("✅ Predictions generated successfully!")
```

```
# Tab layout
tab1, tab2, tab3, tab4, tab5 = st.tabs(["📊 Overview", "📈 Visualizations", "🔍 Analysis", "📤 Export", "⚙ Model Details"])

with tab1:
    st.subheader("Prediction Overview")

    # Metrics cards
    if 'load' in df.columns:
        cols = st.columns(5)
        with cols[0]:
            st.markdown(f"""
            <div class="metric-card">
                <div class="metric-title">R² Score</div>
                <div class="metric-value {'positive' if metrics['R² Score'] > 0.7 else 'negative'}">{metrics['R² Score']:.3f}</div>
                <div style="font-size: 0.8rem; color: {'#28a745' if metrics['R² Score'] > 0.7 else '#dc3545'}">
                    {'Excellent' if metrics['R² Score'] > 0.9 else 'Good' if metrics['R² Score'] > 0.7 else 'Needs Improvement'}
                </div>
            </div>
            """, unsafe_allow_html=True)
        with cols[1]:
            st.markdown(f"""
            <div class="metric-card">
                <div class="metric-title">MAE</div>
                <div class="metric-value">{metrics['MAE']:.2f}</div>
                <div style="font-size: 0.8rem; color: #6c757d">Mean Absolute Error</div>
            </div>
            """, unsafe_allow_html=True)
        with cols[2]:
```

```
                st.markdown(f"""
                <div class="metric-card">
                    <div class="metric-title">RMSE</div>
                    <div class="metric-value">{metrics['RMSE']:.2f}</div>
                    <div style="font-size: 0.8rem; color: #6c757d">Root Mean Squared Error</div>
                </div>
                """, unsafe_allow_html=True)
            with cols[3]:
                st.markdown(f"""
                <div class="metric-card">
                    <div class="metric-title">MAPE</div>
                    <div class="metric-value {'positive' if metrics['MAPE'] < 10 else 'negative'}">{metrics['MAPE']:.1f}%</div>
                    <div style="font-size: 0.8rem; color: {'#28a745' if metrics['MAPE'] < 10 else '#dc3545'}">
                        {'Excellent' if metrics['MAPE'] < 5 else 'Good' if metrics['MAPE'] < 10 else 'Needs Improvement'}
                    </div>
                </div>
                """, unsafe_allow_html=True)
            with cols[4]:
                st.markdown(f"""
                <div class="metric-card">
                    <div class="metric-title">Max Error</div>
                    <div class="metric-value">{metrics['Max Error']:.2f}</div>
                    <div style="font-size: 0.8rem; color: #6c757d">Worst Prediction</div>
                </div>
                """, unsafe_allow_html=True)

        # Data preview
        st.subheader("📋 Prediction Results")
```

```
        show_cols = ['datetime', 'load', 'Predicted Load'] if 'load' in df.columns else ['datetime',
'Predicted Load']


        # Add error column if actual values exist
        if 'load' in df.columns:
            df['Error'] = df['load'] - df['Predicted Load']
            df['Error %'] = (df['Error'] / df['load']) * 100
            show_cols.extend(['Error', 'Error %'])


        # Format the dataframe display
        formatted_df = df[show_cols].copy()
        if 'Error %' in formatted_df.columns:
            formatted_df['Error %'] = formatted_df['Error %'].map("{:.2f}%".format)
        if 'Error' in formatted_df.columns:
            formatted_df['Error'] = formatted_df['Error'].map("{:.2f}".format)
        if 'Predicted Load' in formatted_df.columns:
            formatted_df['Predicted Load'] = formatted_df['Predicted Load'].map("{:.2f}".format)
        if 'load' in formatted_df.columns:
            formatted_df['load'] = formatted_df['load'].map("{:.2f}".format)


        st.dataframe(
            formatted_df.style.apply(
                lambda x: ['background: #e8f5e9' if float(x['Error %'].replace('%','')) < 5 else
                        'background: #ffebee' if float(x['Error %'].replace('%','')) > 10 else ''
                        for i, x in formatted_df.iterrows()],
                axis=1
            ) if 'Error %' in formatted_df.columns else formatted_df,
            use_container_width=True,
            height=400
        )
```

```
        # Add summary statistics
        st.subheader("📊 Summary Statistics")
        if 'load' in df.columns:
            stats_col1, stats_col2 = st.columns(2)

            with stats_col1:
                st.markdown("##### Actual Load")
                st.dataframe(df['load'].describe().to_frame().T.style.format("{:.2f}"),
use_container_width=True)

            with stats_col2:
                st.markdown("##### Predicted Load")
                st.dataframe(df['Predicted Load'].describe().to_frame().T.style.format("{:.2f}"),
use_container_width=True)

    with tab2:
        st.subheader("📈 Interactive Visualizations")

        if 'load' in df.columns:
            # Line plot - Actual vs Predicted
            fig1 = go.Figure()
            fig1.add_trace(go.Scatter(
                x=df['datetime'],
                y=df['load'],
                name='Actual Load',
                line=dict(color='#4a6bff', width=2),
                opacity=0.8
            ))
            fig1.add_trace(go.Scatter(
                x=df['datetime'],
                y=df['Predicted Load'],
                name='Predicted Load',
```

```
    line=dict(color='#ff6b6b', width=2),
    opacity=0.8
))

# Add confidence interval
if confidence_interval > 0:
    std_dev = df['Predicted Load'].std()
    z_score = {80: 1.28, 90: 1.645, 95: 1.96, 99: 2.576}.get(confidence_interval, 1.96)
    margin_of_error = z_score * std_dev

    fig1.add_trace(go.Scatter(
        x=df['datetime'],
        y=df['Predicted Load'] + margin_of_error,
        fill=None,
        mode='lines',
        line=dict(width=0),
        showlegend=False,
        name=f'Upper {confidence_interval}% CI'
    ))

    fig1.add_trace(go.Scatter(
        x=df['datetime'],
        y=df['Predicted Load'] - margin_of_error,
        fill='tonexty',
        mode='lines',
        line=dict(width=0),
        fillcolor='rgba(255, 107, 107, 0.2)',
        showlegend=False,
        name=f'Lower {confidence_interval}% CI'
    ))
```

```
fig1.update_layout(
    title="Actual vs Predicted Load Over Time",
    xaxis_title="Datetime",
    yaxis_title="Load (MW)",
    hovermode="x unified",
    plot_bgcolor='rgba(0,0,0,0)',
    paper_bgcolor='rgba(0,0,0,0)',
    height=600,
    legend=dict(
        orientation="h",
        yanchor="bottom",
        y=1.02,
        xanchor="right",
        x=1
    )
)

# Add range slider
fig1.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1, label="1d", step="day", stepmode="backward"),
                dict(count=7, label="1w", step="day", stepmode="backward"),
                dict(count=1, label="1m", step="month", stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(visible=True),
        type="date"
    )
```

```
)

st.plotly_chart(fig1, use_container_width=True)

# Scatter plot of actual vs predicted
fig_scatter = go.Figure()

fig_scatter.add_trace(go.Scatter(
    x=df['load'],
    y=df['Predicted Load'],
    mode='markers',
    marker=dict(
        color='#4a6bff',
        size=8,
        opacity=0.6
    ),
    name='Predictions'
))

# Add perfect prediction line
max_val = max(df['load'].max(), df['Predicted Load'].max())
min_val = min(df['load'].min(), df['Predicted Load'].min())

fig_scatter.add_trace(go.Scatter(
    x=[min_val, max_val],
    y=[min_val, max_val],
    mode='lines',
    line=dict(color='#ff6b6b', dash='dash'),
    name='Perfect Prediction'
))
```

```
    fig_scatter.update_layout(
        title="Actual vs Predicted Values",
        xaxis_title="Actual Load (MW)",
        yaxis_title="Predicted Load (MW)",
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)',
        height=500,
        showlegend=True
    )

    st.plotly_chart(fig_scatter, use_container_width=True)

# Future forecast visualization
st.subheader("🔮 Future Forecast")

if 'datetime' in df.columns:
    last_date = df['datetime'].max()
    future_dates = [last_date + timedelta(hours=x) for x in range(1, forecast_horizon+1)]

    # For demo purposes - in reality you'd generate future predictions
    future_df = pd.DataFrame({
        'datetime': future_dates,
        'Predicted Load': df['Predicted Load'].tail(forecast_horizon).values
    })

    fig3 = go.Figure()
    fig3.add_trace(go.Scatter(
        x=future_df['datetime'],
        y=future_df['Predicted Load'],
        name='Forecasted Load',
        line=dict(color='#6bff6b', width=3),
```

```
    mode='lines+markers'
))

# Add confidence interval for future forecast
if confidence_interval > 0:
    std_dev = future_df['Predicted Load'].std()
    z_score = {80: 1.28, 90: 1.645, 95: 1.96, 99: 2.576}.get(confidence_interval, 1.96)
    margin_of_error = z_score * std_dev

    fig3.add_trace(go.Scatter(
        x=future_df['datetime'],
        y=future_df['Predicted Load'] + margin_of_error,
        fill=None,
        mode='lines',
        line=dict(width=0),
        showlegend=False,
        name=f'Upper {confidence_interval}% CI'
    ))

    fig3.add_trace(go.Scatter(
        x=future_df['datetime'],
        y=future_df['Predicted Load'] - margin_of_error,
        fill='tonexty',
        mode='lines',
        line=dict(width=0),
        fillcolor='rgba(107, 255, 107, 0.2)',
        showlegend=False,
        name=f'Lower {confidence_interval}% CI'
    ))

fig3.update_layout(
```

```
            title=f"Next {forecast_horizon} Hours Forecast",
            xaxis_title="Datetime",
            yaxis_title="Load (MW)",
            plot_bgcolor='rgba(0,0,0,0)',
            paper_bgcolor='rgba(0,0,0,0)',
            height=500
        )

        st.plotly_chart(fig3, use_container_width=True)

with tab3:
    st.subheader("🔍 Detailed Analysis")

    if 'load' in df.columns:
        # Error distribution
        df['Error'] = df['load'] - df['Predicted Load']
        df['Absolute Error'] = np.abs(df['Error'])
        df['Error Percentage'] = (df['Absolute Error'] / df['load']) * 100

        # Error distribution plot
        st.plotly_chart(plot_error_distribution(df), use_container_width=True)

        # Error by time period
        st.subheader("Error Analysis by Time Period")

        period_col1, period_col2, period_col3 = st.columns(3)

        with period_col1:
            st.markdown("##### By Hour of Day")
            error_by_hour = df.groupby('hour')['Absolute Error'].mean().reset_index()
            fig_hour = go.Figure()
```

```
    fig_hour.add_trace(go.Bar(
        x=error_by_hour['hour'],
        y=error_by_hour['Absolute Error'],
        marker_color='#4a6bff'
    ))
    fig_hour.update_layout(
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)',
        height=300
    )
    st.plotly_chart(fig_hour, use_container_width=True)

with period_col2:
    st.markdown("##### By Day of Week")
    error_by_day = df.groupby('dayofweek')['Absolute Error'].mean().reset_index()
    fig_day = go.Figure()
    fig_day.add_trace(go.Bar(
        x=['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'],
        y=error_by_day['Absolute Error'],
        marker_color='#ff6b6b'
    ))
    fig_day.update_layout(
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)',
        height=300
    )
    st.plotly_chart(fig_day, use_container_width=True)

with period_col3:
    st.markdown("##### By Month")
    error_by_month = df.groupby('month')['Absolute Error'].mean().reset_index()
```

```
            fig_month = go.Figure()
            fig_month.add_trace(go.Bar(
                x=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],
                y=error_by_month['Absolute Error'],
                marker_color='#6bff6b'
            ))
            fig_month.update_layout(
                plot_bgcolor='rgba(0,0,0,0)',
                paper_bgcolor='rgba(0,0,0,0)',
                height=300
            )
            st.plotly_chart(fig_month, use_container_width=True)

        # Worst predictions
        st.subheader("● Top 10 Prediction Errors")
        worst_predictions = df.nlargest(10, 'Absolute Error')[['datetime', 'load', 'Predicted Load', 'Error', 'Error Percentage']]
        st.dataframe(
            worst_predictions.style.format({
                'load': '{:.2f}',
                'Predicted Load': '{:.2f}',
                'Error': '{:.2f}',
                'Error Percentage': '{:.1f}%'
            }).apply(
                lambda x: ['background: #ffebee' if abs(x['Error Percentage']) > 10 else '' for i, x in worst_predictions.iterrows()],
                axis=1
            ),
            use_container_width=True
        )

    with tab4:
```

```
st.subheader("📤 Export Results")

# Report customization
st.markdown("### 📝 Report Customization")
report_title = st.text_input("Report Title", "Load Forecasting Report")

col1, col2 = st.columns(2)
with col1:
    include_metrics = st.checkbox("Include Performance Metrics", True)
    include_charts = st.checkbox("Include Summary Charts", True)
with col2:
    include_raw_data = st.checkbox("Include Raw Data", True)
    include_analysis = st.checkbox("Include Error Analysis", True)

# Download options
st.markdown("### 💾 Download Options")
dl_col1, dl_col2 = st.columns(2)

with dl_col1:
    csv = df.to_csv(index=False).encode('utf-8')
    st.download_button(
        "📥 Download CSV",
        csv,
        "load_forecast_results.csv",
        "text/csv",
        key='download-csv',
        help="Download the complete forecast results as CSV"
    )

with dl_col2:
    # PDF report button (placeholder - would need report generation logic)
```

```
            st.button(
                "📄 Generate PDF Report",
                key='generate-pdf',
                help="Generate a comprehensive PDF report"
            )

        # Email report
        st.markdown("### 📧 Email Report")
        email = st.text_input("Enter recipient email address:", placeholder="user@example.com")

        if st.button("✉ Send Report via Email", key='send-email'):
            if email:
                with st.spinner("Sending email..."):
                    if send_email(email, df, report_title, include_metrics, include_charts):
                        st.success("✅ Report sent successfully!")
                    else:
                        st.error("Failed to send email")
            else:
                st.warning("Please enter a valid email address")

    with tab5:
        st.subheader("⚙ Model Details")

        # Model information
        st.markdown(f"### {selected_model} Model Information")

        if selected_model == "XGBoost":
            st.markdown("""
            *XGBoost (Extreme Gradient Boosting)* is an optimized distributed gradient boosting
library
```

```
            designed to be highly efficient, flexible and portable. It implements machine learning algorithms
            under the Gradient Boosting framework.

            *Advantages:*
            - Handles missing values automatically
            - Regularization helps reduce overfitting
            - Parallel processing for faster training
            - Built-in cross validation
            """)
        elif selected_model == "Random Forest":
            st.markdown("""
            *Random Forest* is an ensemble learning method that operates by constructing a multitude of
            decision trees at training time and outputting the class that is the mode of the classes
            (classification) or mean prediction (regression) of the individual trees.

            *Advantages:*
            - Robust to outliers and noise
            - Handles high dimensional spaces well
            - Provides feature importance measures
            - Less prone to overfitting than single decision trees
            """)
        elif selected_model == "LSTM":
            st.markdown("""
            *LSTM (Long Short-Term Memory)* is a type of recurrent neural network (RNN) architecture that
            is well-suited for time series forecasting tasks. It can learn long-term dependencies in time series data.

            *Advantages:*
            - Excellent for sequence prediction problems
```

```
            - Can learn patterns over varying time scales
            - Robust to noise in input data
            - Can handle multivariate time series
            """)

        # Feature importance visualization
        st.markdown("### 🔍 Feature Importance")
        feature_fig = plot_feature_importance(model, required_columns)
        if feature_fig:
            st.plotly_chart(feature_fig, use_container_width=True)
        else:
            st.info("Feature importance not available for this model type.")

        # Model parameters
        st.markdown("### ⚙ Model Parameters")
        try:
            if hasattr(model, 'get_params'):
                params = model.get_params()
                param_df = pd.DataFrame.from_dict(params, orient='index', columns=['Value'])
                st.dataframe(param_df, use_container_width=True)
            else:
                st.info("Detailed parameters not available for this model.")
        except:
            st.info("Could not retrieve model parameters.")

    except Exception as e:
        st.error(f"❌ An error occurred: {str(e)}")
else:
    # Show demo data and instructions when no file is uploaded
    demo_col1, demo_col2 = st.columns(2)
```

```
    with demo_col1:
        st.markdown("""
        <div class="info-box">
            <h3>📋 How to Use This App</h3>
            <ol>
                <li>Upload your historical load data in CSV format</li>
                <li>Select your preferred forecasting model</li>
                <li>Adjust forecast settings as needed</li>
                <li>View and analyze the predictions</li>
                <li>Export or share your results</li>
            </ol>
            <p>For best results, ensure your data includes datetime and load columns along with relevant features.</p>
        </div>
        """, unsafe_allow_html=True)

    with demo_col2:
        st.markdown("""
        <div class="info-box">
            <h3>📊 Sample Data Format</h3>
            <p>Your CSV file should include these columns (at minimum):</p>
            <ul>
                <li><strong>datetime</strong>: Timestamp for each observation</li>
                <li><strong>load</strong>: Actual electricity load values</li>
                <li><strong>hour</strong>: Hour of day (0-23)</li>
                <li><strong>dayofweek</strong>: Day of week (0-6)</li>
                <li><strong>lag features</strong>: Previous load values (lag_1, lag_24, etc.)</li>
                <li><strong>rolling features</strong>: Rolling averages and std devs</li>
            </ul>
        </div>
        """, unsafe_allow_html=True)
```

```
# Show sample visualization
st.markdown("## 🎯 Sample Forecast Visualization")

# Generate sample data
sample_dates = pd.date_range(start="2023-01-01", periods=24*7, freq="H")
sample_actual = np.sin(np.linspace(0, 10, 24*7)) * 50 + 100 + np.random.normal(0, 5, 24*7)
sample_predicted = np.sin(np.linspace(0, 10, 24*7)) * 50 + 100 + np.random.normal(0, 3, 24*7)

fig_sample = go.Figure()
fig_sample.add_trace(go.Scatter(
    x=sample_dates,
    y=sample_actual,
    name='Actual Load',
    line=dict(color='#4a6bff', width=2),
    opacity=0.8
))
fig_sample.add_trace(go.Scatter(
    x=sample_dates,
    y=sample_predicted,
    name='Predicted Load',
    line=dict(color='#ff6b6b', width=2),
    opacity=0.8
))
fig_sample.update_layout(
    title="Sample Load Forecast (1 Week)",
    xaxis_title="Datetime",
    yaxis_title="Load (MW)",
    hovermode="x unified",
    plot_bgcolor='rgba(0,0,0,0)',
    paper_bgcolor='rgba(0,0,0,0)',
```

```
        height=500
    )
    st.plotly_chart(fig_sample, use_container_width=True)

# ---------- Footer ----------
st.markdown("---")
st.markdown("""
<div style="text-align: center; color: #777; font-size: 0.9rem; padding: 1rem 0;">
    <div style="display: flex; justify-content: center; gap: 1rem; margin-bottom: 0.5rem;">
        <a href="#" style="color: var(--primary); text-decoration: none;">Terms of Service</a>
        <span>•</span>
        <a href="#" style="color: var(--primary); text-decoration: none;">Privacy Policy</a>
        <span>•</span>
        <a href="#" style="color: var(--primary); text-decoration: none;">Documentation</a>
    </div>
    © 2023 Smart Load Forecaster | Powered by AI/ML
</div>
""", unsafe_allow_html=True
```