Project: "90 Days into the Future" - Demand Forecasting

Goal:

To create and assess various ML models for SKU-wise demand forecasting, analyze sales patterns, and predict sales requirements over the next 90 days. We will explore different models and techniques, focusing on accurate prediction and avoiding common pitfalls like skewness and local minima.

Methodology:

Step 1: Data Loading and Preprocessing

```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, PowerTransformer
from fbprophet import Prophet

# Load your sales data (assuming it's in a CSV file)
data = pd.read_csv('sales_data.csv')

# Convert sale date to datetime
data['Sale Date'] = pd.to_datetime(data['Sale Date'])

# Aggregate sales data by SKU and Date
sku_sales = data.groupby(['SKU', 'Sale Date']).agg({'Quantity Sold': 'sum'}).reset_index()

# Feature Engineering: extract date-related features
sku_sales['Year'] = sku_sales['Sale Date'].dt.year
```

sku_sales['Month'] = sku_sales['Sale Date'].dt.month

sku_sales['Day'] = sku_sales['Sale Date'].dt.day

sku_sales['Weekday'] = sku_sales['Sale Date'].dt.weekday

```

## Step 2: Train and Test Split

```python
# Split data into features and target
X = sku_sales[['Year', 'Month', 'Day', 'Weekday']]
y = sku_sales['Quantity Sold']


# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 3: Modeling and Comparison

We will compare the following models:

- **Prophet:** For time-series forecasting.
- **Random Forest Regressor**: For SKU-specific feature importance and prediction.
- **XGBoost**: A powerful boosting algorithm for regression.
- **Linear Regression**: As a baseline model.

Prophet Time Series Model

```python
# Prepare Prophet data for time-series forecasting
prophet_data = sku_sales[['Sale Date', 'Quantity Sold']]
prophet_data.columns = ['ds', 'y']  # Prophet expects 'ds' and 'y' as column names


# Train-Test Split for Prophet (based on date)
train_data = prophet_data[prophet_data['ds'] < '2023-01-01']

```
test_data = prophet_data[prophet_data['ds'] >= '2023-01-01']

# Train the Prophet model
prophet_model = Prophet()
prophet_model.fit(train_data)

# Make future predictions (90 days into the future)
future = prophet_model.make_future_dataframe(periods=90)
forecast = prophet_model.predict(future)

# Plot the forecast
prophet_model.plot(forecast)
```

Random Forest Regressor

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Initialize and train the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test data
y_pred_rf = rf_model.predict(X_test)

# Calculate Mean Squared Error for Random Forest
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f'Random Forest MSE: {mse_rf}')
```

```

## XGBoost Regressor

```python
import xgboost as xgb
from sklearn.metrics import mean_squared_error

# Initialize XGBoost Regressor
xgboost_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Train the model
xgboost_model.fit(X_train, y_train)

# Predict on the test data
y_pred_xgb = xgboost_model.predict(X_test)

# Calculate Mean Squared Error for XGBoost
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
print(f'XGBoost MSE: {mse_xgb}')
```

Linear Regression

```python
from sklearn.linear_model import LinearRegression

# Initialize and train the Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict on the test data

y_pred_lr = lr_model.predict(X_test)

**# Calculate Mean Squared Error for Linear Regression**

mse_lr = mean_squared_error(y_test, y_pred_lr)

print(f'Linear Regression MSE: {mse_lr}')

```

Step 4: Avoiding Local Minima, Skewness, and Improving Training

Tips to Avoid Local Minima and Maxima:

1. **Use Different Optimizers**:

   - Try using adaptive optimizers like **Adam** or **RMSProp**, as they help to avoid local minima by adjusting the learning rate dynamically.

2. **Weight Initialization**:

   - Use **He initialization** for ReLU-based models or **Xavier initialization** for sigmoid/tanh activation functions.

3. **Stochastic Gradient Descent (SGD):**

   - Using **SGD** with momentum can help to escape local minima by allowing the model to "overshoot" them.

Reducing Skewness:

1. **Power Transformation**: Use a **Box-Cox** or **Yeo-Johnson** transformation to stabilize variance and reduce skewness in the data.

```python
from sklearn.preprocessing import PowerTransformer

pt = PowerTransformer()

y_train_transformed = pt.fit_transform(y_train.values.reshape(-1, 1))

y_test_transformed = pt.transform(y_test.values.reshape(-1, 1))
```

2. **Log Transformation**: Apply log transformation to target variables if they are highly skewed.
```

Handling Overfitting:

1. **Cross-Validation**: Use **K-fold cross-validation** to prevent overfitting by testing the model on different subsets of data.

```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(rf_model, X, y, cv=5)
print(f'Cross-Validation Scores: {scores}')
```

2. **Regularization**: Apply **L2 (Ridge)** or **L1 (Lasso)** regularization to penalize large coefficients, which can help generalize the model better.

Effective Data Training:

1. **Standardization**:

- Normalize or scale the data using **StandardScaler** to ensure features are on the same scale.

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

2. **Feature Selection**:

- Perform feature selection to remove irrelevant features, using methods like **Recursive Feature Elimination (RFE)** or feature importance from tree-based models.

```python
from sklearn.feature_selection import RFE
rfe = RFE(rf_model, n_features_to_select=5)
X_train_rfe = rfe.fit_transform(X_train_scaled, y_train)
```

Step 5: Final Accuracy Assessment

Once you've trained the models and predicted results, compare them using:

- **Mean Squared Error (MSE)**: The lower, the better.

- **R-Squared (R^2)**: Higher values indicate better model performance.

```python
from sklearn.metrics import r2_score

r2_rf = r2_score(y_test, y_pred_rf)

r2_xgb = r2_score(y_test, y_pred_xgb)

r2_lr = r2_score(y_test, y_pred_lr)


print(f'Random Forest R²: {r2_rf}')

print(f'XGBoost R²: {r2_xgb}')

print(f'Linear Regression R²: {r2_lr}')
```

This comprehensive approach helps you assess different models, prevent common ML pitfalls, and ensures robust demand forecasting for "90 Days into the Future."

Bibliography

1. **Pandas Documentation:** For data loading, cleaning, and manipulation.

   - pandas.pydata.org/pandas-docs/stable/

2. **Scikit-learn Documentation**: For machine learning model training, evaluation, and preprocessing.

   - scikit-learn.org/stable/

3. **XGBoost Documentation**: For understanding the XGBoost algorithm and its application in regression models.

   - xgboost.readthedocs.io/en/latest/

4. **Prophet Documentation**: Time-series forecasting framework used to predict SKU demand.

   - facebook.github.io/prophet/docs/quick_start.html

5. **Box-Cox and Power Transformation Techniques**: For reducing skewness in data distributions.

- J. D. Johnston, “Transformations of Data,” Biometrics, 14(3), pp. 638–645, 1958.

6. **Gradient Descent Algorithms**: Techniques for optimizing machine learning models, avoiding local minima and maxima.

- S. Ruder, “An overview of gradient descent optimization algorithms,” arXiv preprint arXiv:1609.04747, 2016.

7. **Regularization Techniques in ML**: For reducing overfitting using L1 (Lasso) and L2 (Ridge) regularization.

- T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning," Springer, 2009.

List of Abbreviations

Abbreviation	Meaning	Relation to Project
SKU	Stock Keeping Unit	Identifies individual products for which we are forecasting demand.
MSE	Mean Squared Error	A metric used to measure the average squared difference between predicted and actual values.
R^2	R-Squared	A statistical measure of how close the data are to the fitted regression model.
RF	Random Forest	An ensemble learning method used for regression in demand forecasting.
XGB	XGBoost	A decision-tree-based ensemble algorithm that uses boosting to improve model accuracy.
LR	Linear Regression	A basic model for predicting demand based on historical data.
SGD	Stochastic Gradient Descent	An optimization method used to minimize the error function during model training.
RFE	Recursive Feature Elimination	A technique used to select important features and improve model accuracy.
Box-Cox	Box-Cox Transformation	A method for transforming data to stabilize variance and reduce skewness.

Adam	Adaptive Moment Estimation	An optimization algorithm that adjusts the learning rate for better convergence.
RMSProp	Root Mean Square Propagation	Another adaptive learning rate optimization algorithm.
C1, C2	Control Lines (Inventory Planning)	Used to trigger inventory production plans when the stock reaches 40% and 60% levels.

Explanation of Abbreviations and their Relation to the Project:

1. **SKU (Stock Keeping Unit)**: Critical for identifying individual products for which we are predicting demand. Each SKU has a unique sales pattern.

2. **MSE (Mean Squared Error)**: Used to evaluate model performance by measuring the average squared difference between actual and predicted values. The lower the MSE, the better the model performance.

3. **R^2 (R-Squared)**: This metric helps assess how well our models are explaining the variance in SKU demand. Higher R^2 values indicate a better fit.

4. **RF (Random Forest)**: A machine learning model used for regression tasks to identify patterns in sales data and predict demand. It uses multiple decision trees to increase accuracy.

5. **XGB (XGBoost)**: A powerful model used to improve forecasting accuracy through gradient boosting, which iteratively improves the predictions.

6. **LR (Linear Regression)**: A baseline model to establish initial demand forecasting accuracy.

7. **SGD (Stochastic Gradient Descent)**: Optimization technique used in training machine learning models, particularly helpful in minimizing loss and avoiding local minima.

8. **RFE (Recursive Feature Elimination)**: Used to refine the model by eliminating less important features and keeping only the ones that significantly impact the outcome.

9. **Box-Cox Transformation**: Applied to reduce skewness in data, making models more accurate by improving normality.

10. **Adam & RMSProp**: These optimizers dynamically adjust learning rates during training to avoid getting stuck in local minima, leading to better convergence.

11. **C1, C2 (Control Lines)**: Part of inventory planning, these levels are used to trigger production decisions based on sales forecasts and consumption patterns, ensuring that the SKU demand is met without overproduction.