DemandCast build plan

Alright — if we're going to make **DemandCast – AI-Based Demand Forecasting for E-commerce Products** your full-fledged project, I'll give you **a single, end-to-end master guide** so you have **everything** you need:

- Technical blueprint
- Tools & libraries
- Data sources
- ML modeling steps
- EDA & feature engineering
- Optimization
- Evaluation
- Deployment
- Documentation
- Industry-level polish

This will be the **detailed playbook** for execution.

📖 DemandCast – Complete Project Guide

1. Project Overview

Goal: Predict future product demand using historical sales, seasonality, social sentiment, and marketing campaign data.
Type: Time Series + Machine Learning + External Data Integration
Real-world impact: Helps e-commerce businesses plan inventory, optimize ads, and reduce overstock/stockouts.

---

2. Project Architecture

```mathematica
Data Collection → Data Preprocessing → EDA & Feature Engineering → Model Training & Validation → Model Selection → Deployment → Monitoring
```

---

3. Data Sources

You'll need **multiple data streams** to make it more realistic:

A. Historical Sales Data

- Columns: `date`, `product_id`, `category`, `units_sold`, `price`, `discount`, `stock_level`
- Get from:
    - Kaggle dataset: "Retail Sales Forecasting" / "Walmart Sales"
    - Or simulate with Faker + seasonality patterns.

B. External Factors

- **Holiday & Event Calendar:** Google public holidays API, or custom CSV.
- **Weather Data:** OpenWeatherMap API (temperature, rainfall, humidity).
- **Social Media Sentiment:** Twitter/X API (brand mentions, product sentiment).
- **Marketing Spend:** Simulated budget per week/month.

---

4. Tools & Libraries

Core Python Stack:

```python
 pandas, numpy, matplotlib, seaborn
scikit-learn, statsmodels
prophet (Facebook Prophet)
xgboost, lightgbm, catboost
tensorflow/keras (for LSTM/GRU)
pmdarima (auto_arima)
nltk, vaderSentiment (for sentiment analysis)
requests, beautifulsoup4 (for scraping)
```

Deployment:

- **Web:** Streamlit / Flask
- **Cloud:** AWS EC2 / Azure App Service / GCP App Engine
- **Database:** PostgreSQL / MongoDB

---

5. Data Pipeline

Step 1: Collection

```python
import pandas as pd
# Load CSV
sales_df = pd.read_csv('sales_data.csv', parse_dates=['date'])

# Example API data fetch
import requests
weather = requests.get("https://api.openweathermap.org/data/...").json()
```

Step 2: Cleaning

- Handle missing values (`fillna`, interpolation).
- Remove anomalies (e.g., negative sales).
- Align timezones & formats.

Step 3: Merge Sources

```python
merged_df = sales_df.merge(weather_df, on='date').merge(sentiment_df, on='date')
```

---

6. Exploratory Data Analysis (EDA)

- Sales trend over time.
- Seasonal patterns (weekly/monthly spikes).
- Correlation between weather/holidays and sales.
- Impact of marketing spend on sales.

Example Visualization:

```python
import matplotlib.pyplot as plt
plt.plot(merged_df['date'], merged_df['units_sold'])
plt.title("Sales Over Time")
```

---

7. Feature Engineering

- **Lag features:** `sales_lag_1`, `sales_lag_7`, `sales_lag_30`
- **Rolling stats:** moving averages, rolling std
- **Seasonal indicators:** `is_weekend`, `is_holiday`
- **External factors:** temperature, sentiment score, ad spend
- **Price elasticity:** percent change in price vs sales

---

8. Modeling

We'll compare **three families** of models:

A. Classical Time Series

- ARIMA / SARIMA
- Facebook Prophet

B. Machine Learning

- XGBoost / LightGBM (with engineered time features)

C. Deep Learning

- LSTM / GRU networks for sequence modeling

Example: Prophet Model

```python
from prophet import Prophet

df_prophet = merged_df[['date', 'units_sold']].rename(columns={'date':'ds', 'units_sold':'y'})
model = Prophet()
model.add_regressor('temperature')
model.add_regressor('sentiment_score')
model.fit(df_prophet)
forecast = model.predict(df_prophet)
```

---

9. Model Evaluation

Metrics to track:

- **RMSE** (Root Mean Squared Error)
- **MAE** (Mean Absolute Error)

- **MAPE** (Mean Absolute Percentage Error)
- Compare performance per model & per category.

---

10. Optimization

- Hyperparameter tuning: `sklearn.model_selection.GridSearchCV` or Optuna.
- Feature importance ranking (XGBoost).
- Model stacking (combine Prophet + XGBoost).

---

11. Deployment

A. Streamlit App

- File upload for new sales data.
- Model forecast for next 30/60/90 days.
- Interactive charts with Plotly.
- Recommendations (e.g., “Increase stock for Product X”).

B. API

- Flask/FastAPI endpoint for predictions.

---

12. Monitoring

- Save model artifacts (`joblib` / `pickle`).
- Schedule daily/weekly retraining.
- Track prediction accuracy drift.

---

13. Deliverables

✅ Source code (GitHub repo)
✅ Requirements file (`requirements.txt`)
✅ Dataset (raw & processed)
✅ Final ML model(s)
✅ Deployment app link
✅ Project report & PPT for presentation

---

14. Extra Polishing for Resume

- Add a **dashboard view** (Streamlit tabs for Sales Trends, Forecast, Feature Importance).
- Include **PDF export** of predictions.
- Show **real-time external data fetch** (like current weather & sentiment).
- Integrate **Google Sheets live sync** for input data.

If you want, I can now **turn this into a full 90-day step-by-step calendar** where each day has a specific subtask — matching your earlier Google Calendar import — so that you literally have a **to-do per day** until completion.
That would make it execution-ready without thinking “what do I do today?”.

Do you want me to create that detailed day-by-day breakdown next?

Create Google Docs Create Word Docs Create PDF Docs

Alright — let’s make this a **master execution blueprint** for your *DemandCast – AI-Based Demand Forecasting for E-commerce Products* so you not only know **what to do,** but also **exactly when to do it, what tools to use, what deliverables to produce, and what checkpoints to hit.**

I’ll make it in a **phased timeline** (Weeks → Days → Tasks), with:

- 📅 **Exact scheduling** so you can execute without confusion.
- 🛠 **Tools & libraries** for each step.
- 📂 **Expected outputs** so you know when you’re “done” with a step.
- ✅ **Checkpoints** to ensure quality before moving on.

📌 Phase 1 – Project Setup & Requirement Gathering (Days 1–4)

Goal: Establish the foundation — understand the problem, set scope, and prep your environment.

Day	Task	Tools	Deliverable	Checkpoint
1	Define business goal: "Predict product	Google Docs / Notion	Project brief	Stakeholder goal clear, metrics decided (MAPE,

Day	Task	Tools	Deliverable	Checkpoint
	demand 1–3 months ahead for inventory optimization"			RMSE)
2	Research E-commerce demand forecasting techniques	Google Scholar, Medium	Research doc	3–4 baseline models shortlisted
3	Decide input features (historical sales, seasonality, holidays, ads, sentiment)	Excel/Google Sheets	Feature list	Approved feature list
4	Set up Python environment	Python 3.11, Conda, VS Code	GitHub repo	Environment works, packages installed

---

📌 Phase 2 – Data Collection & Storage (Days 5–10)

Goal: Gather, store, and preprocess raw data.

Day	Task	Tools	Deliverable	Checkpoint
5	Collect historical sales data	CSV exports / API	`sales_data.csv`	Data covers min 2 years
6	Collect holidays/calendar data	Python `holidays` lib	`holidays.csv`	Holiday mapping ready
7	Gather marketing/ad campaign data	Marketing dashboard/API	`ad_spend.csv`	Synced to sales timeline
8	Pull product pricing history	API / Web scraping	`price_history.csv`	Clean format

Day	Task	Tools	Deliverable	Checkpoint
9	Collect social sentiment (Twitter, Reddit)	Tweepy, PRAW, Vader/BERT	`sentiment.csv`	Sentiment scores ready
10	Store all datasets in database	PostgreSQL/MongoDB	Central DB	All datasets accessible via SQL/NoSQL

📌 Phase 3 – Data Preprocessing & EDA (Days 11–17)

Goal: Clean, transform, and understand the data.

Day	Task	Tools	Deliverable	Checkpoint
11–12	Data cleaning (missing values, outliers)	Pandas, NumPy	Clean datasets	No NaNs, no duplicates
13–14	Feature engineering (lags, moving averages, seasonal indicators)	Pandas, FeatureTools	New feature CSV	At least 10 engineered features
15	Exploratory Data Analysis (EDA)	Matplotlib, Seaborn	EDA notebook	All trends visualized
16	Correlation & seasonality analysis	Statsmodels, Pandas	Seasonal plots	Seasonality confirmed
17	Train/test split (time-series split)	scikit-learn	Split datasets	No data leakage

---

📌 Phase 4 – Baseline Model Building (Days 18–25)

Goal: Create initial models for benchmarking.

Day	Task	Tools	Deliverable	Checkpoint
18–19	ARIMA/SARIMA	Statsmodels	`arima_model.pkl`	Residuals stationary

Day	Task	Tools	Deliverable	Checkpoint
20	Prophet model	Facebook Prophet	`prophet_model.pkl`	Good fit on validation
21–22	LSTM time-series	TensorFlow/Keras	`lstm_model.h5`	Trains without overfit
23	XGBoost regression	XGBoost	`xgb_model.pkl`	Baseline results stored
24	Compare models (MAPE, RMSE)	Pandas	Comparison table	Best baseline chosen
25	Document baseline results	Notion/Google Docs	Baseline report	Ready for stakeholders

---

📌 Phase 5 – Advanced Model Tuning (Days 26–35)

Goal: Optimize and improve models.

Day	Task	Tools	Deliverable	Checkpoint
26–27	Hyperparameter tuning (Optuna/GridSearchCV)	Optuna, scikit-learn	Tuned params	MAPE improved
28	Feature importance & selection	SHAP, Permutation Importance	Feature ranking	Low-importance features removed
29–30	Ensemble models (Stacking, Blending)	scikit-learn, LightGBM	Ensemble model	Outperforms baselines
31–33	Model retraining with tuned params	TensorFlow, Prophet, XGB	Final models	Consistent results
34–35	Final evaluation on hold-out data	Pandas	Final metrics report	Metrics stable

---

📌 Phase 6 – Deployment & Dashboard (Days 36–45)

Goal: Make the solution accessible.

Day	Task	Tools	Deliverable	Checkpoint
36	Choose deployment stack	Streamlit/Flask/FastAPI	Tech decision doc	Tech stack locked
37–39	Build API endpoints	FastAPI/Flask	API code	Can return forecast JSON
40–42	Create dashboard UI	Streamlit	Interactive dashboard	Displays forecasts
43	Connect DB + live model	SQLAlchemy/PyMongo	Integrated app	Works end-to-end
44	Testing & bug fixing	pytest, Postman	Bug-free app	Passes all tests
45	Deploy to cloud	Heroku/AWS	Live URL	Accessible publicly

📌 Phase 7 – Documentation, Testing & Presentation (Days 46–50)

Goal: Final polish and handover.

Day	Task	Tools	Deliverable	Checkpoint
46–47	Final project documentation	Markdown, Notion	README.md	Fully documented
48	Prepare project presentation	PowerPoint	Slide deck	Tells project story
49	Dry run presentation	Google Meet	Feedback notes	Ready for review
50	Final submission	GitHub, Drive	Project package	Delivered successfully

---

🔹 Total Timeline Overview

- **Phase 1–2:** Days 1–10 → Planning & Data Collection
- **Phase 3:** Days 11–17 → Data Cleaning & EDA
- **Phase 4:** Days 18–25 → Baseline Models
- **Phase 5:** Days 26–35 → Advanced Tuning
- **Phase 6:** Days 36–45 → Deployment & Dashboard
- **Phase 7:** Days 46–50 → Documentation & Presentation

If you want, I can now **convert this into an ICS file** so each task is an actual Google Calendar event **with reminders and deadlines**, so you'll get daily alerts.

Do you want me to prepare that next?

Create Google Docs Create Word Docs Create PDF Docs