1. Explain about problem statement
    a. Must predict whether a flight will depart with a delay of 15 minutes or more using only pre-departure operational and weather information,
2. Explain about `data_directory.ipynb`
    a. Explain purpose
    b. How I have done it
3. 01_flight_data_preprocessing.ipynb
    a. Explain purpose – In order to integrate flight and weather data, we have know the structure data first. So started with flight data, I took a 2016 Jan data for initial exploration.
    b. Since we only worry about on pre-dep delays, based on the domain knowledge, I have selected the features which satisfy one of the rules.
    c. Explore
        i. data,
        ii. checked for target distribution,
        iii. behavior of CRSDEPTIME,
        iv. converted FLIGHTDATE from str to date format,
        v. Airports data & filtered flights supported to weather-data
        vi. Checked for missing data here
        vii. Cancelled, diverted vs Missing Delayed flights
        viii. Handled missing data
        ix. Load & combine the flight data
4. 02_weather_data_preprocessing. ipynb
    a. Like flight data, explore the weather data as well. But here we have JSON format.
    b. Since some hourly weather is nested, So I Flattened the data
    c. Ran some validation checks like duplicates, missing values etc
    d. Flatten & combine all weather data
    e. Final validation check on final combined dataset
5. 03_merge_flight_weather_data.ipynb
    a. Creating matching keys to join both data
    b. Created dep_hour → hhmm format to 0-23 hours bucket
    c. Quality check before merging
    d. Performed merge with left join with flight data
    e. Check merge success rate and inspected the rows
    f. Validation checks & save the the final dataset
6. 04_exploratory_data_analysis.ipynb
    a. EDA on final weather dataset

b. Load data
c. Target class distribution – majority of flights are not delayed, shows class imbalance
d. Time of the day vs Delays
    i. Delay by hour – highest delay in the evenings
    ii. Flight volume – its not volume driven
    iii. Feature Importance & Non-linearity – DEP_HOUR strong predictor & target is non-linear.
    iv. LR → partially capture the trends, so Tree-based models are suited
e. Day-of-Week Effects on Delays
    i. Same as above time-of-day
    ii. It effects moderate, not dominant
f. Route & Distance vs Delays
    i. BY ORG → large hub airport have high delay rates, smaller show more stable operations with fewer delays → are important factors & influence the likelihood of delays
    ii. BY DIST_GROUP → Looks relatively weak but might be effective when combined with time, airport & weather features
g. Unique Carrier vs Delays
    i. To find out whether a particular airline choice alone affects DEP DELAY
    ii. Shows some carriers consistently higher delay rates than others around 3times
    iii. To go beyond visualization, I quantified the between-carrier variance using a weighted squared deviation approach.
    iv. The resulting effect size was about 0.00136, which indicates that airline choice explains a measurable portion of total delay variability — independent of weather and time
h. Carrier Effect Vs Time of the Day
    i. After identifying that delay rates differ significantly by airline, I wanted to verify that this wasn't simply because some carriers operate more flights during peak congestion hours.
    ii. So, I computed delay probability per airline per departure hour and measured the standard deviation across hours.
    iii. Some airlines show high hour-to-hour variability, while others remain consistent. Importantly, carrier differences persist even after accounting for departure hour.

        iv. This confirms that airline choice provides independent predictive value beyond scheduling effects.
    i. Weather vs Delay – checking with precipitation, wind, visibility, cloud cover
        i. Precipitation → Sharp increase once it begins, after moderate rain operations already degraded
        ii. Wind Speed→ High wind speed – high departure delays
        iii. Visibility → Usually severe low-visibility often leads to cancellations, not delays, Does not act as a simple linear predictor of delays
        iv. Cloud Cover → gradual increase in delay, weaker than precipitation & winds
        v. Weather conditions show a strong relationship with departure delays,
        vi. They contribute predictive power through non-linear & interaction effects, favoring tree-based modeling approaches.
    j. Feature Correlations & Interactions
        i. Among numeric features - **CRSDepTime & dep_hour, Distance & DistanceGroup, tempC & FeelsLikeC, windspeedKmph & windgustKmph,** → we drop one of among each other
        ii. **humidity & cloudcover (Positive), precipMM & cloudcover (Positive), visibility & humidity (Negative), pressure & precipMM (Negative)** → aligns with real world cases
    k. Correlation with target
        i. They are non-linear, conditional, interaction driven
        ii. Delays are driven by a combination of temporal, operational & weather-related factors rather than any single dominant variable
    l. Interaction: Time of day & Day of the week
        i. Morning flights are on time
        ii. Friday is darker than others
    m. Key findings
7. 05_baseline_modeling_data.ipynb
    a. Problem Framing
        i. Define prediction target – DepDel15
    b. Evaluation metrics on imbalanced data
        i. Accuracy can be misleading
        ii. roc-auc, precision/recall, f1 score
        iii. Recall > precision is preferred for operations teams to ensure readiness
    c. Establish a Baseline Model

    i. Logistic Regression as a baseline model to establish minimum viable performance

d. Based on the EDA, we have remove the redundant features- cancelled, diverted, CRSDepTime, Distance, FeelsLikeC, Year
e. Some for just LR – DepTimeBlk, winddirDegree, weathercode
f. Environment setup & data loading
g. Defined the target & feature variables
h. Splitting the data – Using Stratification
    i. There is a reason behind this, since the dataset covered only three months and this project was focused on model development rather than deployment, I used a stratified random split to ensure stable class distribution and sufficient training signal.
    ii. A strict temporal split would have reduced the effective training window significantly. However, in a production setting, I would absolutely switch to a temporal or rolling window validation approach to properly simulate forward prediction and evaluate performance stability over time.

i. Logistic Regression
    i. I used Logistic Regression as a baseline to establish whether meaningful predictive signal exists in the pre-departure features. Since this is a binary classification problem — predicting whether a flight will be delayed by 15 minutes or more — logistic regression provides a simple and interpretable starting point.
    ii. I built a preprocessing pipeline where numeric features were scaled using StandardScaler, and categorical features like carrier and airport were one-hot encoded. Everything was wrapped inside a Pipeline to prevent data leakage and ensure consistent transformations between training and testing.
    iii. Because only about 20% of flights are delayed, I used class_weight="balanced" to adjust for class imbalance. This encouraged the model to identify delayed flights more aggressively.
    iv. The model achieved a ROC-AUC of about 0.66, indicating moderate discrimination. Recall for delayed flights was around 62%, meaning it successfully identified more than half of actual delays. However, precision was relatively low, suggesting many false positives.
    v. Overall, the baseline confirmed that predictive signal exists, but performance limitations — particularly moderate AUC and low precision — indicated that delay patterns are likely driven by non-

linear interactions. That insight motivated the transition to tree-based ensemble models.

j. Random Forest
    i. I moved to Random Forest after establishing the Logistic Regression baseline, since the baseline results suggested that delay patterns were not linearly separable. EDA showed strong interaction effects between departure hour and day of week, as well as non-linear threshold behavior in weather variables. Random Forest is well-suited for capturing these complex, non-linear relationships.
    ii. For preprocessing, I retained one-hot encoding for categorical variables such as carrier and airport. However, I removed feature scaling for numeric variables because tree-based models are invariant to feature magnitude and do not require normalization. As before, the preprocessing and model were wrapped inside a Pipeline to ensure consistent transformations and prevent leakage.
    iii. I configured the Random Forest with 300 trees to stabilize predictions, limited the maximum depth to prevent overfitting, and set a minimum number of samples per leaf to smooth splits. I also retained class_weight="balanced" to maintain sensitivity toward delayed flights.
    iv. The Random Forest improved ROC-AUC from approximately 0.66 to 0.72, indicating significantly better class discrimination. Recall for delayed flights remained strong at around 61%, while precision improved slightly compared to Logistic Regression. This confirms that allowing the model to capture non-linear interactions improves separation between delayed and on-time flights.
    v. Overall, the improvement in AUC validated the hypothesis from EDA that delay behavior is interaction-driven and non-linear. While performance improved meaningfully, there was still room to optimize the precision–recall tradeoff, which motivated exploring boosting-based models in subsequent steps.

k. Comparision
    i. Random Forest improved ROC-AUC from 0.66 to 0.72 and PR-AUC from 0.33 to 0.42, confirming better discrimination under class imbalance. Precision improved while recall remained stable, increasing overall F1-score.

ii. This improvement is consistent with EDA findings that delay behavior is non-linear and interaction-driven — patterns that tree-based models naturally capture better than linear models.

l. Feature Importance
    i. After I just want to check which feature importance in the RF.
    ii. Feature importance shows that departure hour is by far the dominant predictor, confirming the delay propagation effect observed in EDA.
    iii. Weather variables like precipitation, wind speed, and pressure follow, indicating non-linear environmental impact on delays. Airport and carrier features also appear, reflecting operational heterogeneity.
    iv. Overall, the importance of ranking aligns strongly with exploratory findings and reinforces that delay behavior is driven by temporal patterns, weather thresholds, and structural airport differences.

m. SHAP
    i. The distribution of SHAP values shows both positive and negative contributions, confirming that delay behavior is interaction-driven and non-linear — which explains why tree-based models outperform linear models.

n. Threshold Tuning & PR Analysis section
    i. Since the dataset is imbalanced, using the default 0.5 probability threshold is not necessarily optimal.
    ii. I generated the precision–recall curve to analyze the trade-off between identifying delayed flights and avoiding false positives.
    iii. As expected, lowering the threshold increases recall but reduces precision; while increasing it improves precision at the cost of recall.
    iv. When evaluating F1 across thresholds, the optimal value coincided with the default 0.5 cutoff, indicating that model limitations stem from class separability rather than poor threshold choice.
    v. This confirms that improving model structure, not just decision rules, was necessary.

8. 06_final_modeling_data.ipynb
    a. Aim to increase recall for delayed flights without compromising precision
    b. Final feature set Definition
    c. Tuned Random Forest
        i. After evaluating the baseline Random Forest, I introduced controlled hyperparameter tuning to improve ranking quality and minority-class discrimination rather than just overall accuracy.

ii. First, I increased **n_estimators to 400** to reduce variance and stabilize ensemble predictions. Increasing tree count lowers model variance by averaging more independently trained trees, improving probability calibration and ranking consistency.
iii. I increased **max_depth to 18** while simultaneously enforcing **min_samples_leaf=50 and min_samples_split=100**. This combination allows the model to capture higher-order feature interactions while preventing overfitting to noise. Essentially, depth increases model capacity, but leaf and split constraints regularize it.
iv. Instead of using oversampling or SMOTE, I applied cost-sensitive learning through **class_weight={0:1, 1:3}.** This modifies the impurity calculation (Gini) to penalize misclassification of the minority class more heavily, directly influencing split decisions at the tree level. This approach preserves the original data distribution and avoids synthetic sample bias, which is particularly important in operational and temporally structured datasets like flight delays.
v. Performance Metrics
vi. This tuning improved the bias–variance trade-off by slightly increasing model complexity while controlling leaf purity, resulting in better minority-class discrimination without introducing synthetic data artifacts

d. **Gradient Boosting - Baseline**
   i. After Random Forest, I transitioned to Gradient Boosting to reduce bias and improve ranking quality through sequential error correction.
   ii. Unlike Random Forest, which builds trees independently and averages them (bagging), Gradient Boosting builds trees sequentially. Each new tree is trained to fit the residual errors of the previous ensemble using gradient descent in function space.
   iii. This allows the model to:
      1. Focus on hard-to-classify examples
      2. Reduce bias progressively
      3. Learn more refined decision boundaries
   iv. Hyperparameter Logic, I configured:
      1. n_estimators=200 → sufficient boosting rounds to learn residual structure
      2. learning_rate=0.05 → smaller step size for smoother optimization

3. max_depth=3 → shallow trees to prevent overfitting and enforce weak learners
4. This follows boosting theory: Many shallow trees + small learning rate = controlled additive modeling.

v. Performance Interpretation
   1. At the default threshold (0.5), recall for delayed flights was very low due to class imbalance.
   2. Instead of retraining, I tuned the decision threshold to 0.30.
   3. Lowering the threshold:
      a. Increased recall significantly (≈ 6% → 34%)
      b. Reduced precision moderately
      c. Maintained ROC-AUC around 0.71
   4. This confirms an important point:
      a. ROC-AUC remained stable because threshold changes do not affect ranking quality only classification cutoff.
      b. So, the underlying discrimination ability of the model did not change.
   5. Compared to Random Forest:
      a. Random Forest reduces variance via bagging
      b. Gradient Boosting reduces bias via sequential correction
   6. In your results:
      a. ROC-AUC ≈ 0.71 (similar to RF baseline)
      b. PR-AUC ≈ 0.41 (slightly lower than tuned RF)
      c. Precision higher, but recall weaker at default threshold
   7. This suggests: Boosting learned sharper boundaries but may need stronger class weighting or custom loss to improve minority recall.

vi. The top features remained:
   1. Departure hour, Precipitation, Calendar variables, Airport-specific effects
   2. This consistency across models strengthens the validity of your findings.
   3. It shows the signal is structural, not model specific.

vii. Gradient Boosting confirmed that the problem benefits from sequential error correction, but class imbalance required threshold tuning. Performance improvements came primarily from ranking quality rather than raw threshold shifts.

e. **Tuned Gradient Boosting**

i. After evaluating the baseline Gradient Boosting model, I introduced structured hyperparameter tuning to improve ranking performance and minority-class discrimination, rather than optimizing for accuracy alone.

ii. First, I tuned the ensemble size and learning dynamics. Increasing n_estimators to 350 while keeping learning_rate=0.1 allowed the model to reduce bias by performing more sequential residual corrections. In Gradient Boosting, each tree fits the negative gradient of the loss function (log-loss for classification), so increasing estimators improves functional approximation of the decision boundary.

iii. However, to prevent variance explosion, I constrained model complexity through max_depth=3 and min_samples_leaf=50. Shallow trees limit high-order interaction complexity, while a large leaf size smooths probability estimates and prevents sharp, noise-driven splits. This combination improves generalization and stabilizes probability calibration.

iv. I also introduced subsample=0.8, which converts standard Gradient Boosting into stochastic gradient boosting. By training each tree on a random 80% subset of the data, the model reduces variance similarly to bagging while maintaining boosting's bias-reduction properties. This adds regularization without sacrificing ranking quality.

v. Importantly, I tuned using scoring="roc_auc" rather than accuracy. ROC-AUC evaluates ranking quality across thresholds, which is critical in imbalanced classification because threshold-dependent metrics like accuracy can be misleading. Since the decision threshold can be adjusted post-training, optimizing the ranking metric ensures the model separates classes effectively in probability space.

vi. From a performance perspective:

1. ROC-AUC improved from ~0.71 to ~0.73, indicating stronger class separability.
2. Recall for delayed flights improved from ~34% to ~40%, which is a meaningful gain under imbalance.
3. F1-score increased from ~0.39 to ~0.43, reflecting better balance between precision and recall.
4. Accuracy remained stable (~0.79), meaning the improvement did not degrade majority-class performance.

vii. The confusion matrix shows an increase in true positives with only a moderate increase in false positives. This suggests improved boundary refinement rather than aggressive positive prediction.

viii. **Final line -** This tuning improved the bias–variance trade-off by increasing functional approximation capacity through additional boosting rounds while controlling variance via structural and stochastic regularization. The resulting model achieved stronger minority-class ranking and discrimination without introducing synthetic data artifacts or distorting the original data distribution.

f. XGBoost – Final Model

i. After evaluating Logistic Regression, Random Forest, and sklearn Gradient Boosting, I moved to XGBoost because:

1. It includes explicit regularization (L1/L2) to reduce overfitting.
2. It handles class imbalance more effectively via scale_pos_weight.
3. It uses second-order gradient optimization (both gradient & Hessian).
4. It has shrinkage + subsampling, improving bias–variance tradeoff.
5. It's computationally optimized and typically improves ROC-AUC by 2–5%.

ii. Handling Class Imbalance

1. Class ratio:

$$scale\_pos\_weight = \frac{negative}{positive} \approx 4.0$$

2. To prevent over-amplifying minority errors, I **softened** it:

$$used\ \text{scale}\_pos\_weight = 2.0$$

3. This stabilizes learning and avoids aggressive overprediction of delays.

iii. Objective: I used

1. objective = "binary:logistic"
   eval_metric = "auc"
2. To optimized for **ranking quality (ROC-AUC)**, not raw accuracy.

iv. Key hyperparameters:

1. n_estimators = 400 → More boosting rounds (reduce bias)
2. learning_rate = 0.05 → Small steps, stable convergence
3. max_depth = 4 → Capture interactions without overfitting

4. subsample = 0.8 → Stochastic boosting (variance reduction)
5. colsample_bytree = 0.8 → Feature subsampling (decorrelation)
6. reg_lambda = 1.0 → L2 regularization
7. scale_pos_weight ≈ 2.0 → Imbalance correction
8. This setup balances model capacity and regularization.

v. Threshold Optimization (Critical Step)
    1. Instead of using the default 0.5 cutoff:
        a. I computed the Precision–Recall curve
        b. Calculated F1 at every threshold
        c. Selected the threshold maximizing F1 (got best ~0.39)
    2. This is important because:
        a. ROC-AUC measures ranking
        b. Threshold controls operational trade-off
        c. Imbalanced problems require threshold tuning

vi. Performance of model, Got:
    1. Highest ROC-AUC → Best ranking power
    2. Highest PR-AUC → Best performance under imbalance
    3. Recall improved from ~34% → 54%
    4. Substantially fewer missed delays
    5. Acceptable trade-off in false positives

vii. This is a structural improvement, not just threshold manipulation.

viii. After selecting XGBoost, I validated model behavior using **SHAP** (TreeExplainer)

ix. Global Drivers (SHAP Mean |Value|)
    1. Top features: Carrier-specific effects (AS, NK), Departure hour, Destination SFO, Weather codes (storm/fog/heavy rain), Temperature & humidity, Airport congestion patterns
    2. This confirms:
        a. Late departures increase delay probability
        b. Adverse weather pushes prediction positive
        c. Certain carriers/airports systematically influence delay risk
    3. The model learned realistic operational relationships.

x. From summary plot:
    1. High departure hour → Positive SHAP → Higher delay probability
    2. Severe weather codes → Positive SHAP
    3. Certain carriers → Consistent directional influence
    4. Calm weather → Negative SHAP (protective effect)

xi. This validates domain alignment.

g. Final Model Selection Justification

i. XGBoost was selected because:

1. Highest ranking performance (ROC-AUC)
2. Strongest imbalance handling (PR-AUC)
3. Best recall for delay detection
4. Stable probability calibration
5. Interpretable via SHAP
6. Regularized and production-ready

ii. It balances:

1. Predictive performance
2. Business relevance (fewer missed delays)
3. Model stability
4. Interpretability

9. **To summarize, this project demonstrates that flight delay prediction is fundamentally a non-linear, interaction-driven problem. By integrating operational schedule data with weather features and applying tree-based models, I achieved a ROC-AUC of 0.735 and captured 54% of delayed flights using a threshold-optimized XGBoost model. More importantly, I validated model behavior using SHAP to ensure interpretability and operational realism. With extended temporal data and upstream delay features, this framework can be production-ready and deliver meaningful value to airline operations**.