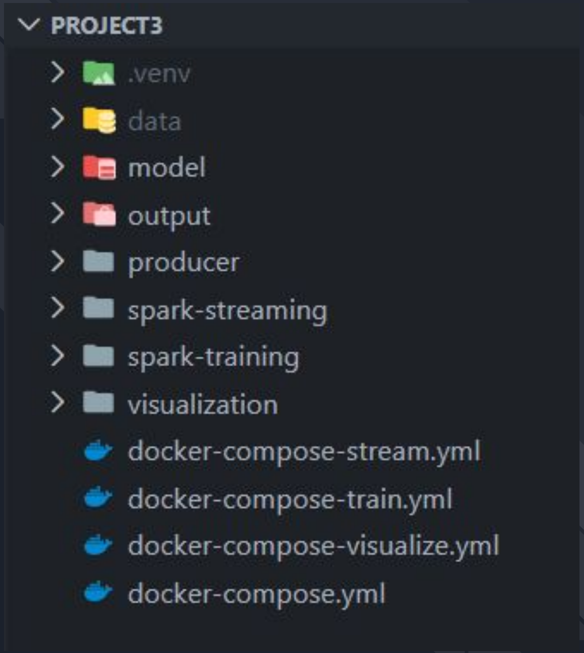
Upravljanje i analiza velikih skupova podataka

Projekat 3 - Spark MLlib

Luka Gavrilović 1823

Projekat 3

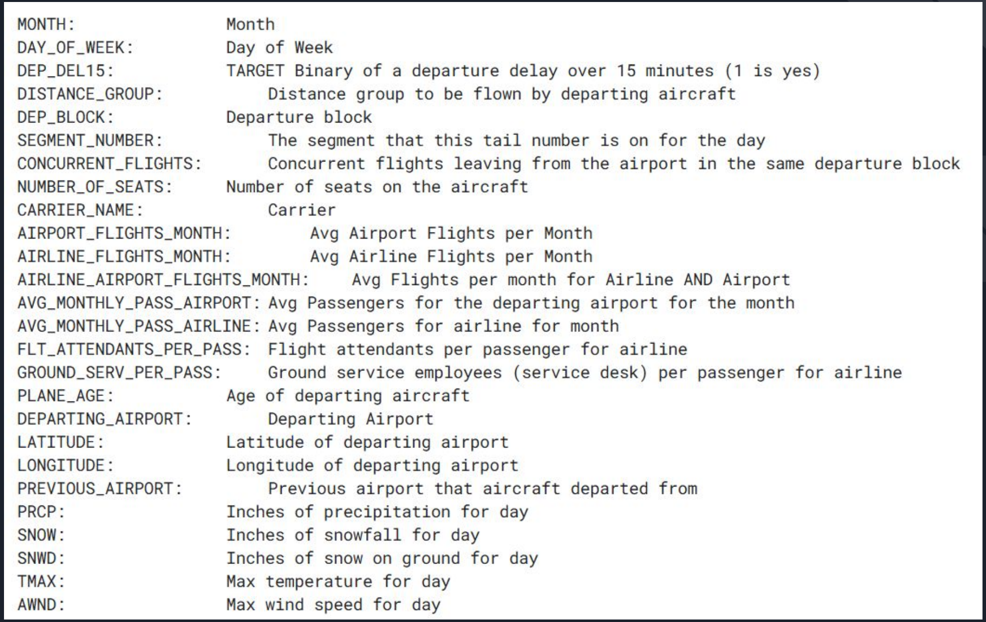
- Real-time predviđanje kašnjenja letova:
  - Treniranje modela za predikciju kašnjenja letova koristeći istorijske podatke
  - Predviđanje kašnjenja u realnom vremenu sa Kafka stream-om
  - Testiranje performansi na Spark Docker klasteru
  - Vizualizacija rezultata po vremenskim prozorima i geografskim lokacijama



```
PROJECT3
  > .venv
  > data
  > model
  > output
  > producer
  > spark-streaming
  > spark-training
  > visualization
    docker-compose-stream.yml
    docker-compose-train.yml
    docker-compose-visualize.yml
    docker-compose.yml
```

Skup podataka

- 2019 Airline Delays w/Weather and Airport Detail (≈1.37GB)
- https://www.kaggle.com/datasets/threnjen/2019-airline-delays-and-cancellations?select=full_data_flightdelay.csv
- Skup podataka sa detaljnim informacijama o avio-kompanijama, aerodromima i vremenskim uslovima



MONTH:	Month
DAY_OF_WEEK:	Day of Week
DEP_DEL15:	TARGET Binary of a departure delay over 15 minutes (1 is yes)
DISTANCE_GROUP:	Distance group to be flown by departing aircraft
DEP_BLOCK:	Departure block
SEGMENT_NUMBER:	The segment that this tail number is on for the day
CONCURRENT_FLIGHTS:	Concurrent flights leaving from the airport in the same departure block
NUMBER_OF_SEATS:	Number of seats on the aircraft
CARRIER_NAME:	Carrier
AIRPORT_FLIGHTS_MONTH:	Avg Airport Flights per Month
AIRLINE_FLIGHTS_MONTH:	Avg Airline Flights per Month
AIRLINE_AIRPORT_FLIGHTS_MONTH:	Avg Flights per month for Airline AND Airport
AVG_MONTHLY_PASS_AIRPORT:	Avg Passengers for the departing airport for the month
AVG_MONTHLY_PASS_AIRLINE:	Avg Passengers for airline for month
FLT_ATTENDANTS_PER_PASS:	Flight attendants per passenger for airline
GROUND_SERV_PER_PASS:	Ground service employees (service desk) per passenger for airline
PLANE_AGE:	Age of departing aircraft
DEPARTING_AIRPORT:	Departing Airport
LATITUDE:	Latitude of departing airport
LONGITUDE:	Longitude of departing airport
PREVIOUS_AIRPORT:	Previous airport that aircraft departed from
PRCP:	Inches of precipitation for day
SNOW:	Inches of snowfall for day
SNWD:	Inches of snow on ground for day
TMAX:	Max temperature for day
AWND:	Max wind speed for day

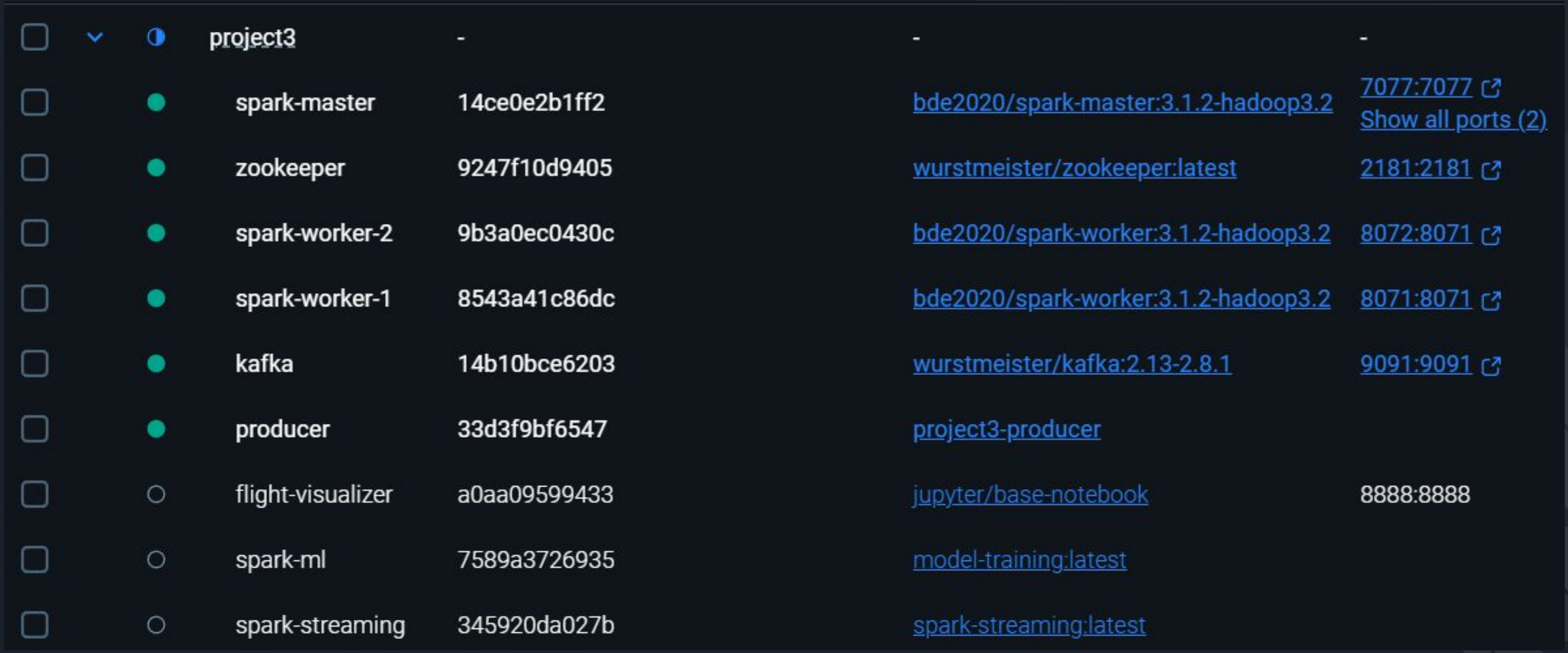
Redosled pokretanja Docker compose servisa

Redosled pokretanja je bitan zato što neki servisi zavise od drugih da bi funkcionisali:

1. `docker network create bde` (ako nije kreirana mreža)
2. `docker-compose up --build -d`
3. `docker-compose -f docker-compose-train.yml up --build`
4. `docker-compose -f docker-compose-stream.yml up --build`
5. `docker-compose -f docker-compose-visualize.yml up --build`

- Moguće je i pokretanje servisa u detached mode-u, odnosno u pozadini
- U tom slučaju logovi se mogu pratiti komandom: `docker logs -f <container_name>`

Docker kontejneri



☐	⌄	◑	project3	-	-	-
☐		●	spark-master	14ce0e2b1ff2	bde2020/spark-master:3.1.2-hadoop3.2	7077:7077 Show all ports (2)
☐		●	zookeeper	9247f10d9405	wurstmeister/zookeeper:latest	2181:2181
☐		●	spark-worker-2	9b3a0ec0430c	bde2020/spark-worker:3.1.2-hadoop3.2	8072:8071
☐		●	spark-worker-1	8543a41c86dc	bde2020/spark-worker:3.1.2-hadoop3.2	8071:8071
☐		●	kafka	14b10bce6203	wurstmeister/kafka:2.13-2.8.1	9091:9091
☐		●	producer	33d3f9bf6547	project3-producer	
☐		○	flight-visualizer	a0aa09599433	jupyter/base-notebook	8888:8888
☐		○	spark-ml	7589a3726935	model-training:latest	
☐		○	spark-streaming	345920da027b	spark-streaming:latest	

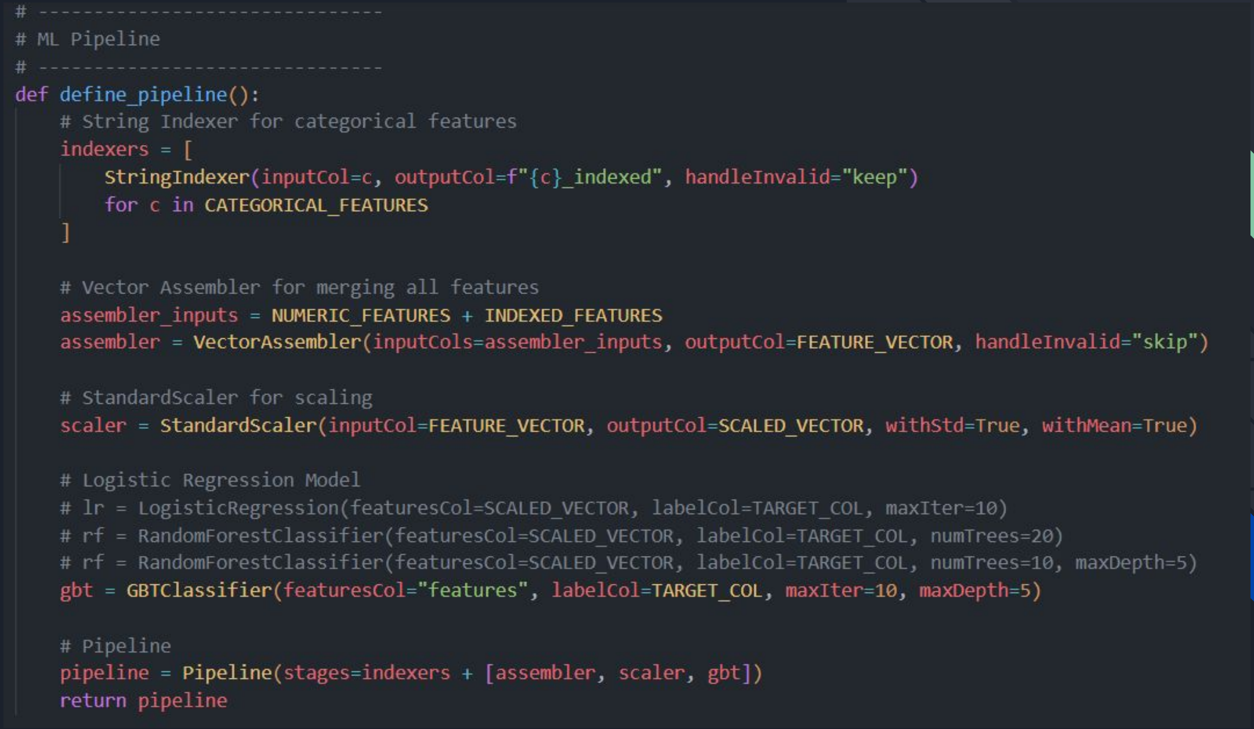
Trening ML modela - Spark MLlib

- Algoritam: GBTClassifier (alternativa: RandomForest, LogisticRegression)
- Pipeline: StringIndexer → VectorAssembler → StandardScaler → GBT
- Evaluacija
- Model sačuvan u `/model/lr_pipeline_model`
- Pokreće se ukoliko ne postoji model u `/model` folderu, ili ako se menja ML model

- Pokretanje offline treninga ML modela:
  - pozicioniranje u folder gde se nalazi `docker-compose-train.yml`
  - `docker-compose -f docker-compose-train.yml up --build`

Trening ML modela - Spark MLlib

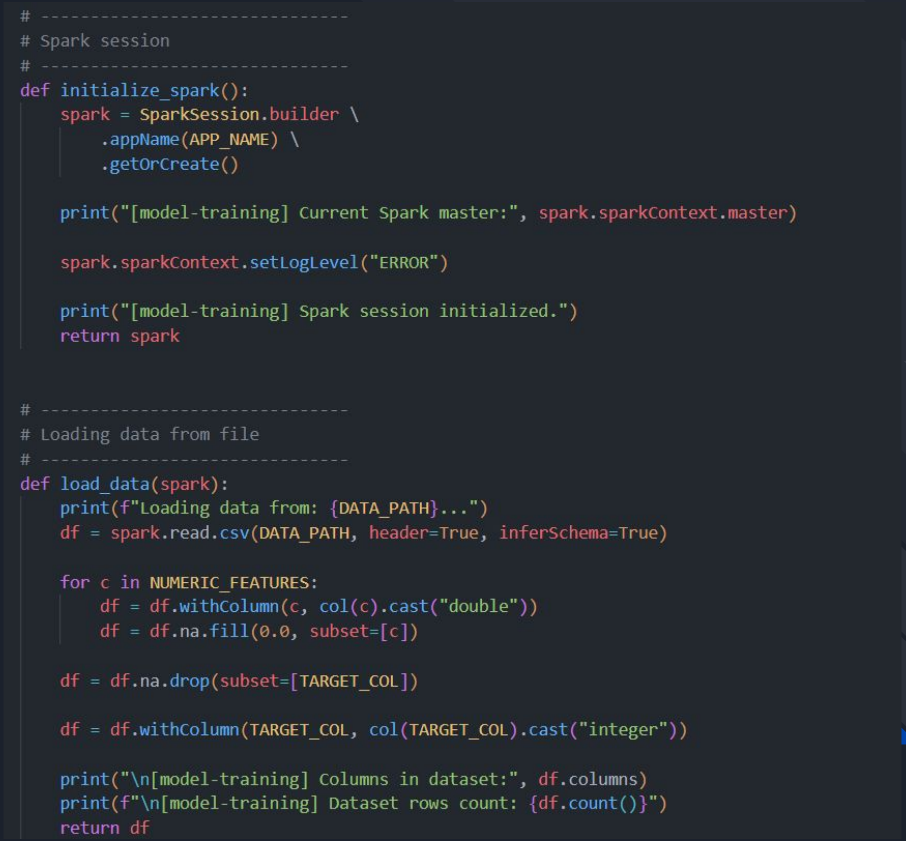
- Kreiranje ML pipeline



```python
# ------------------------------
# ML Pipeline
# ------------------------------
def define_pipeline():
    # String Indexer for categorical features
    indexers = [
        StringIndexer(inputCol=c, outputCol=f"{c}_indexed", handleInvalid="keep")
        for c in CATEGORICAL_FEATURES
    ]

    # Vector Assembler for merging all features
    assembler_inputs = NUMERIC_FEATURES + INDEXED_FEATURES
    assembler = VectorAssembler(inputCols=assembler_inputs, outputCol=FEATURE_VECTOR, handleInvalid="skip")

    # StandardScaler for scaling
    scaler = StandardScaler(inputCol=FEATURE_VECTOR, outputCol=SCALED_VECTOR, withStd=True, withMean=True)

    # Logistic Regression Model
    # lr = LogisticRegression(featuresCol=SCALED_VECTOR, labelCol=TARGET_COL, maxIter=10)
    # rf = RandomForestClassifier(featuresCol=SCALED_VECTOR, labelCol=TARGET_COL, numTrees=20)
    # rf = RandomForestClassifier(featuresCol=SCALED_VECTOR, labelCol=TARGET_COL, numTrees=10, maxDepth=5)
    gbt = GBTClassifier(featuresCol="features", labelCol=TARGET_COL, maxIter=10, maxDepth=5)

    # Pipeline
    pipeline = Pipeline(stages=indexers + [assembler, scaler, gbt])
    return pipeline
```

Trening ML modela - Spark MLlib

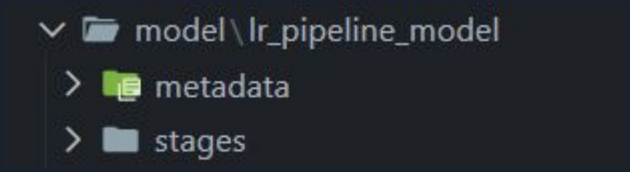
- Spark inicijalizacija i učitavanje podataka

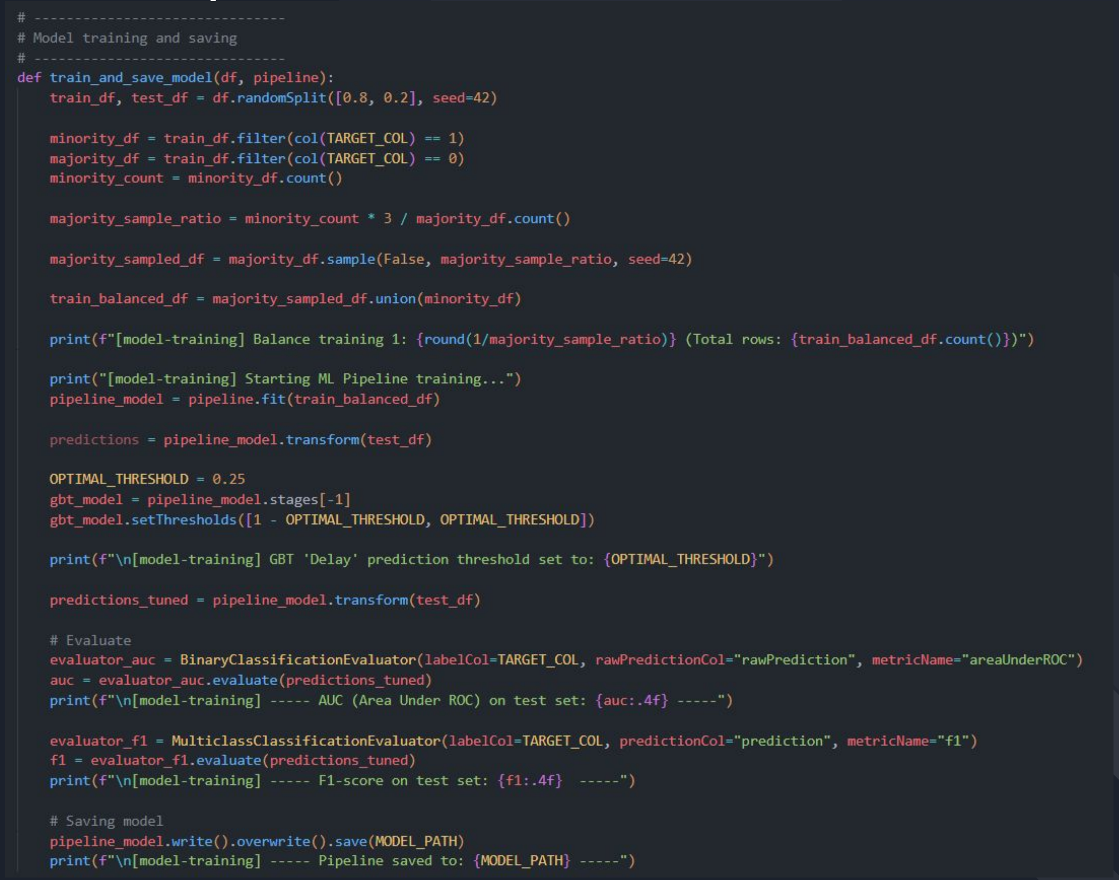


```
# ------------------------------
# Spark session
# ------------------------------
def initialize_spark():
    spark = SparkSession.builder \
        .appName(APP_NAME) \
        .getOrCreate()

    print("[model-training] Current Spark master:", spark.sparkContext.master)

    spark.sparkContext.setLogLevel("ERROR")

    print("[model-training] Spark session initialized.")
    return spark


# ------------------------------
# Loading data from file
# ------------------------------
def load_data(spark):
    print(f"Loading data from: {DATA_PATH}...")
    df = spark.read.csv(DATA_PATH, header=True, inferSchema=True)

    for c in NUMERIC_FEATURES:
        df = df.withColumn(c, col(c).cast("double"))
        df = df.na.fill(0.0, subset=[c])

    df = df.na.drop(subset=[TARGET_COL])

    df = df.withColumn(TARGET_COL, col(TARGET_COL).cast("integer"))

    print("\n[model-training] Columns in dataset:", df.columns)
    print(f"\n[model-training] Dataset rows count: {df.count()}")
    return df
```

Trening ML modela - Spark MLlib

- Treniranje modela i čuvanje u MODEL_PATH (`/model/lr_pipeline_model`)

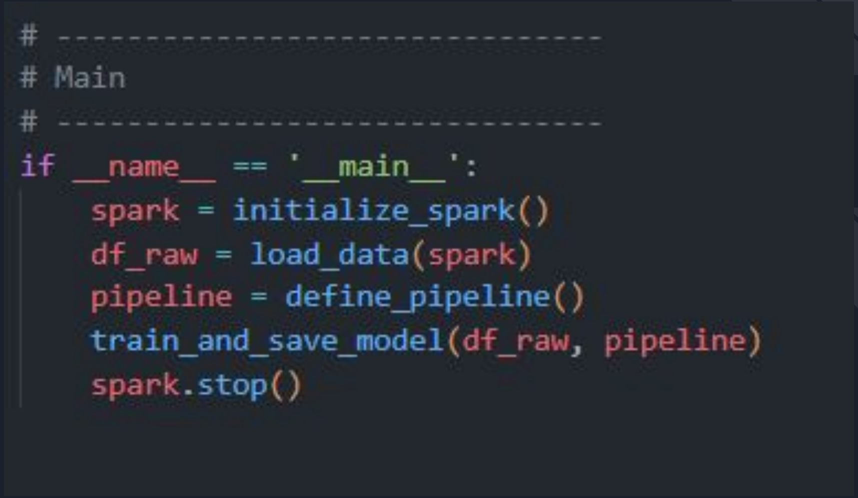




```
# -------------------------------
# Model training and saving
# -------------------------------
def train_and_save_model(df, pipeline):
    train_df, test_df = df.randomSplit([0.8, 0.2], seed=42)

    minority_df = train_df.filter(col(TARGET_COL) == 1)
    majority_df = train_df.filter(col(TARGET_COL) == 0)
    minority_count = minority_df.count()

    majority_sample_ratio = minority_count * 3 / majority_df.count()

    majority_sampled_df = majority_df.sample(False, majority_sample_ratio, seed=42)

    train_balanced_df = majority_sampled_df.union(minority_df)

    print(f"[model-training] Balance training 1: {round(1/majority_sample_ratio)} (Total rows: {train_balanced_df.count()})")

    print("[model-training] Starting ML Pipeline training...")
    pipeline_model = pipeline.fit(train_balanced_df)

    predictions = pipeline_model.transform(test_df)

    OPTIMAL_THRESHOLD = 0.25
    gbt_model = pipeline_model.stages[-1]
    gbt_model.setThresholds([1 - OPTIMAL_THRESHOLD, OPTIMAL_THRESHOLD])

    print(f"\n[model-training] GBT 'Delay' prediction threshold set to: {OPTIMAL_THRESHOLD}")

    predictions_tuned = pipeline_model.transform(test_df)

    # Evaluate
    evaluator_auc = BinaryClassificationEvaluator(labelCol=TARGET_COL, rawPredictionCol="rawPrediction", metricName="areaUnderROC")
    auc = evaluator_auc.evaluate(predictions_tuned)
    print(f"\n[model-training] ----- AUC (Area Under ROC) on test set: {auc:.4f} -----")

    evaluator_f1 = MulticlassClassificationEvaluator(labelCol=TARGET_COL, predictionCol="prediction", metricName="f1")
    f1 = evaluator_f1.evaluate(predictions_tuned)
    print(f"\n[model-training] ----- F1-score on test set: {f1:.4f}  -----")

    # Saving model
    pipeline_model.write().overwrite().save(MODEL_PATH)
    print(f"\n[model-training] ----- Pipeline saved to: {MODEL_PATH} -----")
```

Trening ML modela - Spark MLlib

- Main



```python
# --------------------------------
# Main
# --------------------------------
if __name__ == '__main__':
    spark = initialize_spark()
    df_raw = load_data(spark)
    pipeline = define_pipeline()
    train_and_save_model(df_raw, pipeline)
    spark.stop()
```

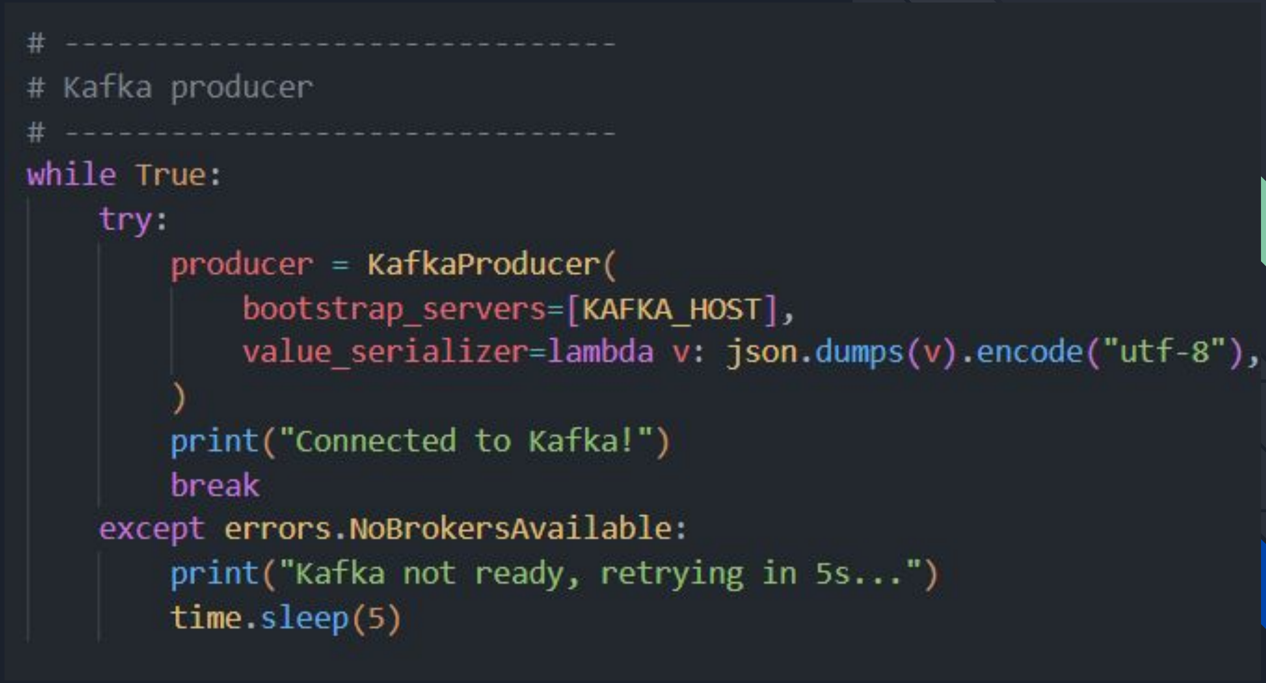
Producer i Spark klaster kontejnera

- `docker-compose` fajl za pokretanje Spark klastera, učitavanja i slanja podataka na Kafka topic
- `docker-compose.yml`klaster konfiguracija:
  - Spark master + 2 workera
  - Kafka & Zookeeper
  - Producer
- Aplikacija pokreće se unutar mreže `bde`
- Kreira se producer koji čita .csv podatke i šalje na Kafka topic

- Pokretanje infrastrukture:
  - pozicioniranje u folder gde se nalazi `docker-compose.yml`
  - `docker-compose up --build -d`

Producer

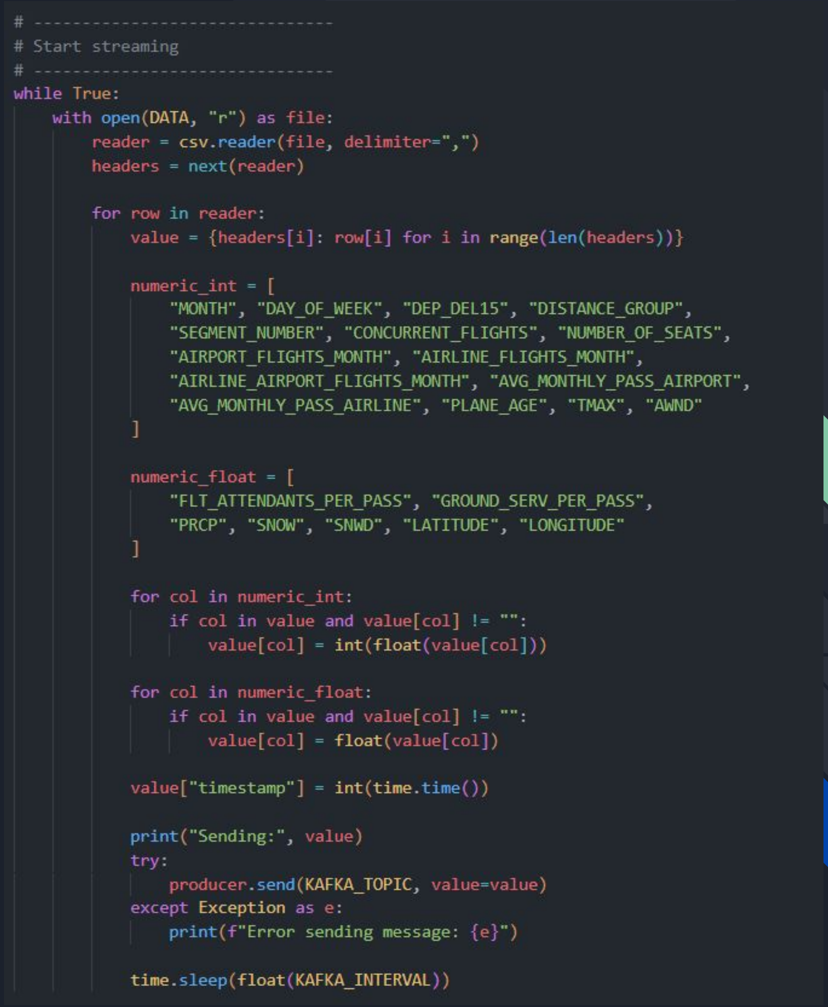
- Konektovanje na Kafku



```
# ------------------------------
# Kafka producer
# ------------------------------
while True:
    try:
        producer = KafkaProducer(
            bootstrap_servers=[KAFKA_HOST],
            value_serializer=lambda v: json.dumps(v).encode("utf-8"),
        )
        print("Connected to Kafka!")
        break
    except errors.NoBrokersAvailable:
        print("Kafka not ready, retrying in 5s...")
        time.sleep(5)
```

Producer

- Slanje podataka na topic KAFKA_TOPIC



```python
# ------------------------------
# Start streaming
# ------------------------------
while True:
    with open(DATA, "r") as file:
        reader = csv.reader(file, delimiter=",")
        headers = next(reader)

        for row in reader:
            value = {headers[i]: row[i] for i in range(len(headers))}

            numeric_int = [
                "MONTH", "DAY_OF_WEEK", "DEP_DEL15", "DISTANCE_GROUP",
                "SEGMENT_NUMBER", "CONCURRENT_FLIGHTS", "NUMBER_OF_SEATS",
                "AIRPORT_FLIGHTS_MONTH", "AIRLINE_FLIGHTS_MONTH",
                "AIRLINE_AIRPORT_FLIGHTS_MONTH", "AVG_MONTHLY_PASS_AIRPORT",
                "AVG_MONTHLY_PASS_AIRLINE", "PLANE_AGE", "TMAX", "AWND"
            ]

            numeric_float = [
                "FLT_ATTENDANTS_PER_PASS", "GROUND_SERV_PER_PASS",
                "PRCP", "SNOW", "SNWD", "LATITUDE", "LONGITUDE"
            ]

            for col in numeric_int:
                if col in value and value[col] != "":
                    value[col] = int(float(value[col]))

            for col in numeric_float:
                if col in value and value[col] != "":
                    value[col] = float(value[col])

            value["timestamp"] = int(time.time())

            print("Sending:", value)
            try:
                producer.send(KAFKA_TOPIC, value=value)
            except Exception as e:
                print(f"Error sending message: {e}")

            time.sleep(float(KAFKA_INTERVAL))
```

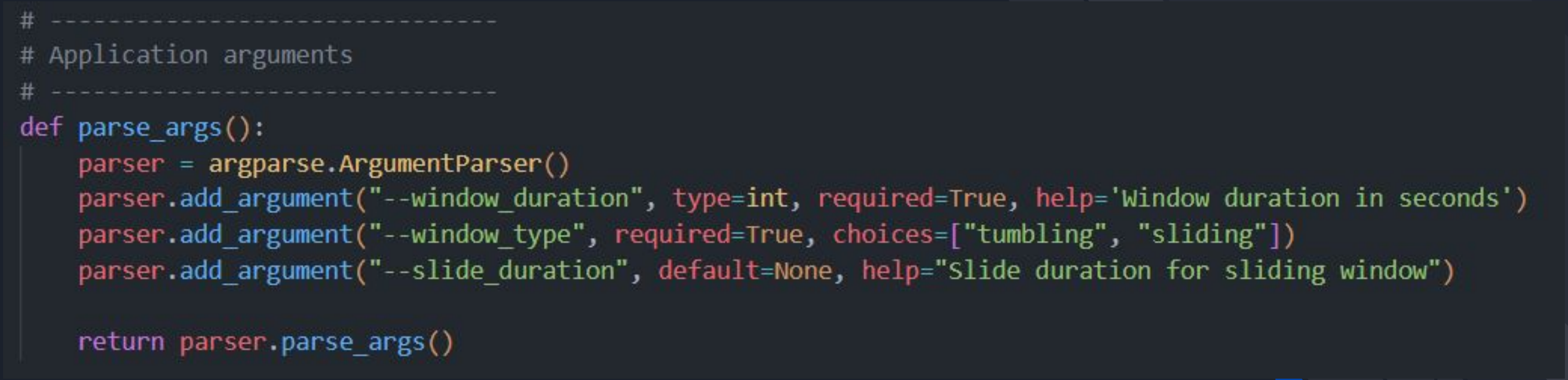
Spark streaming

- Učitava poruke sa Kafka topic-a
- Primjenjuje prethodno trenirani ML model
- Agregira podatke po prozorima
- Čuva rezultate u `/output/predictions`u CSV formatu

- Pokretanje Spark streaming aplikacije:
  - pozicioniranje u folder gde se nalazi `docker-compose-stream.yml`
  - `docker-compose -f docker-compose-stream.yml up --build`

Spark streaming

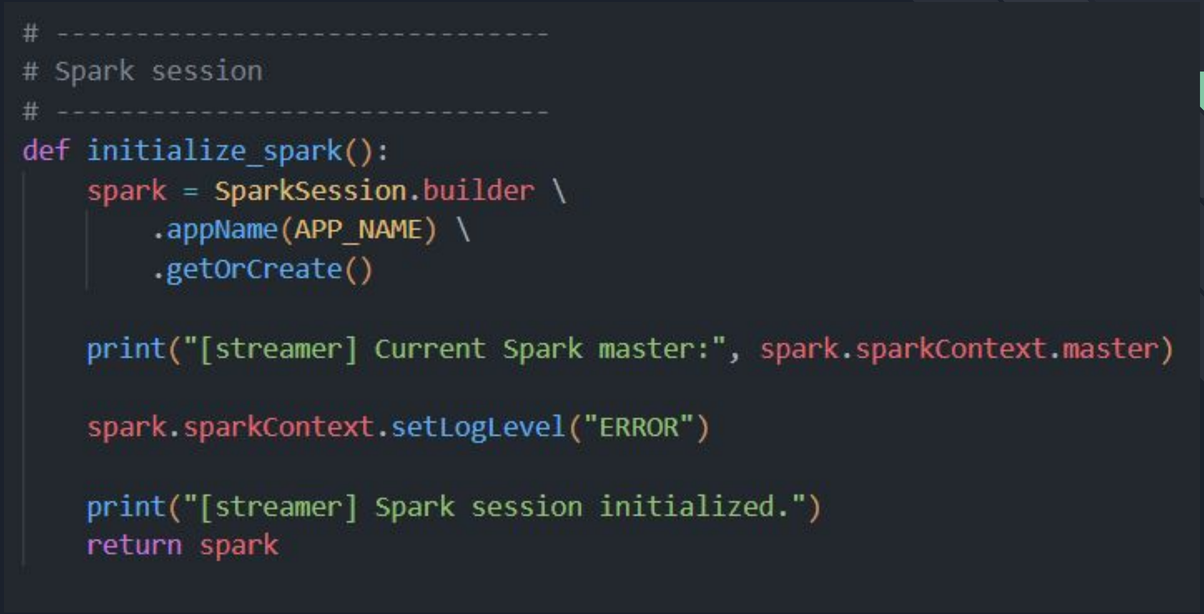
- Argumenti aplikacije



```
# ------------------------------
# Application arguments
# ------------------------------
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--window_duration", type=int, required=True, help='Window duration in seconds')
    parser.add_argument("--window_type", required=True, choices=["tumbling", "sliding"])
    parser.add_argument("--slide_duration", default=None, help="Slide duration for sliding window")

    return parser.parse_args()
```

Spark streaming

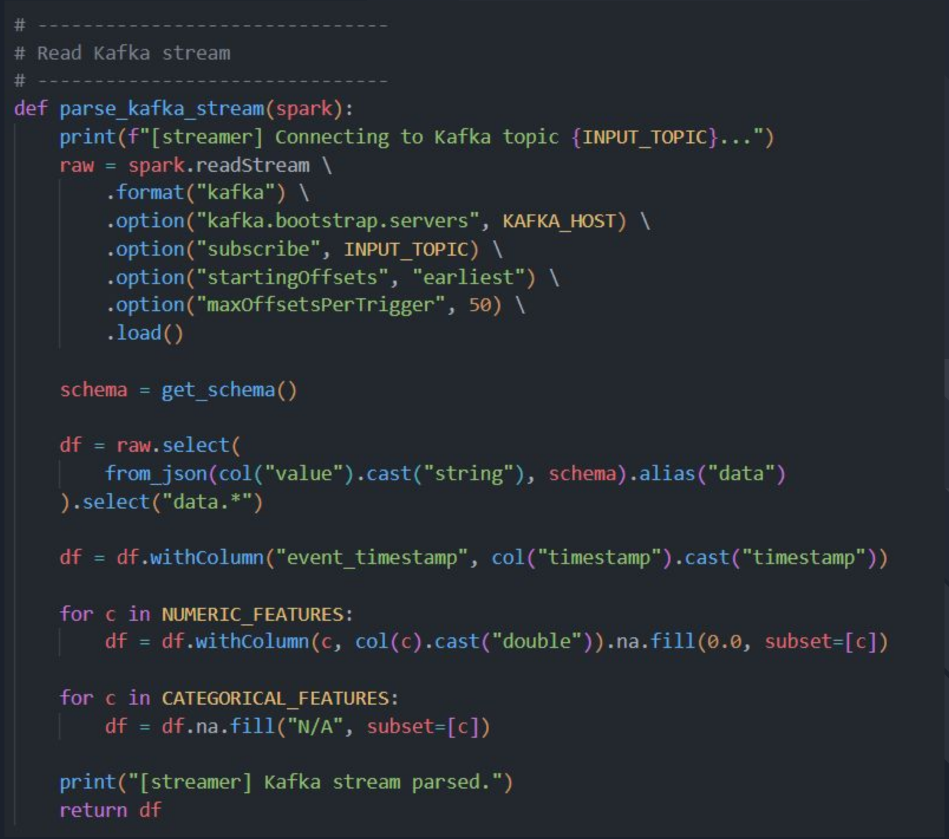
- Spark inicijalizacija



```python
# ------------------------------
# Spark session
# ------------------------------
def initialize_spark():
    spark = SparkSession.builder \
        .appName(APP_NAME) \
        .getOrCreate()

    print("[streamer] Current Spark master:", spark.sparkContext.master)

    spark.sparkContext.setLogLevel("ERROR")

    print("[streamer] Spark session initialized.")
    return spark
```

Spark streaming

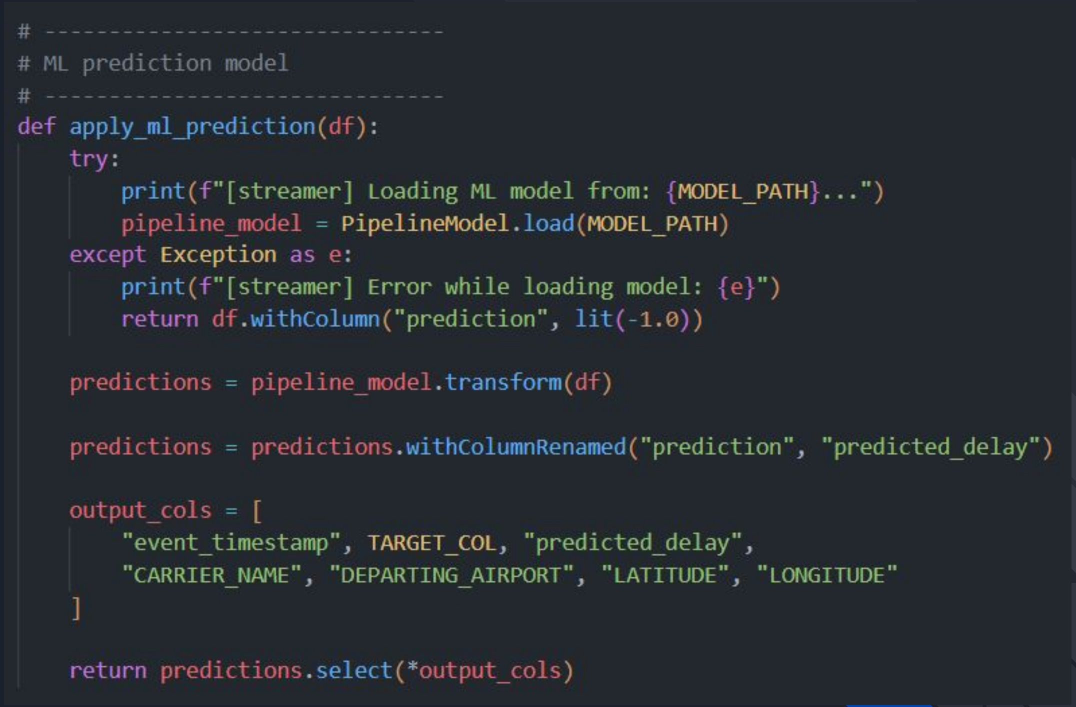
- Čitanje Kafka stream-a



```python
# ------------------------------
# Read Kafka stream
# ------------------------------
def parse_kafka_stream(spark):
    print(f"[streamer] Connecting to Kafka topic {INPUT_TOPIC}...")
    raw = spark.readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", KAFKA_HOST) \
        .option("subscribe", INPUT_TOPIC) \
        .option("startingOffsets", "earliest") \
        .option("maxOffsetsPerTrigger", 50) \
        .load()

    schema = get_schema()

    df = raw.select(
        from_json(col("value").cast("string"), schema).alias("data")
    ).select("data.*")

    df = df.withColumn("event_timestamp", col("timestamp").cast("timestamp"))

    for c in NUMERIC_FEATURES:
        df = df.withColumn(c, col(c).cast("double")).na.fill(0.0, subset=[c])

    for c in CATEGORICAL_FEATURES:
        df = df.na.fill("N/A", subset=[c])

    print("[streamer] Kafka stream parsed.")
    return df
```

Spark streaming

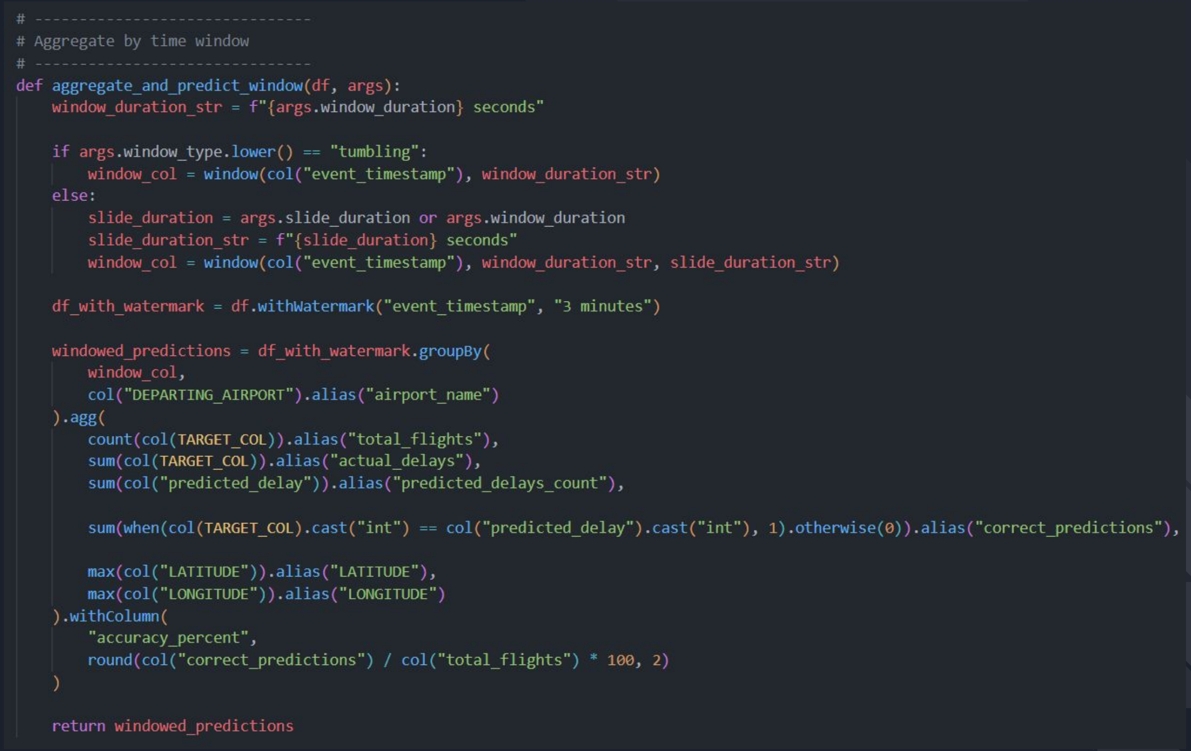
- Primena ML modela na učitane podatke sa Kafka topic-a



```python
# ------------------------------
# ML prediction model
# ------------------------------
def apply_ml_prediction(df):
    try:
        print(f"[streamer] Loading ML model from: {MODEL_PATH}...")
        pipeline_model = PipelineModel.load(MODEL_PATH)
    except Exception as e:
        print(f"[streamer] Error while loading model: {e}")
        return df.withColumn("prediction", lit(-1.0))

    predictions = pipeline_model.transform(df)

    predictions = predictions.withColumnRenamed("prediction", "predicted_delay")

    output_cols = [
        "event_timestamp", TARGET_COL, "predicted_delay",
        "CARRIER_NAME", "DEPARTING_AIRPORT", "LATITUDE", "LONGITUDE"
    ]

    return predictions.select(*output_cols)
```

Spark streaming

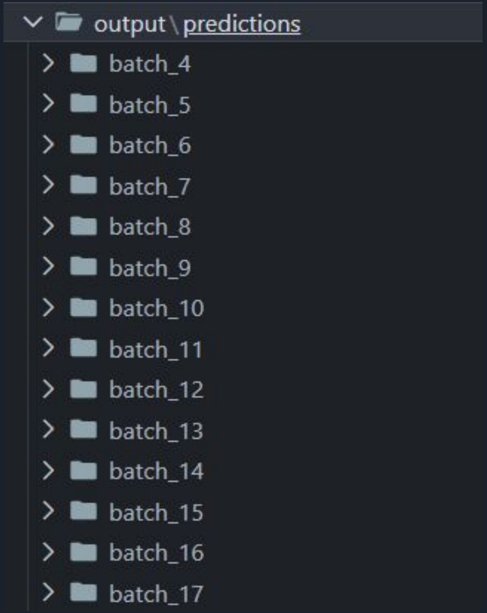
- Agregacija podataka po prozorima



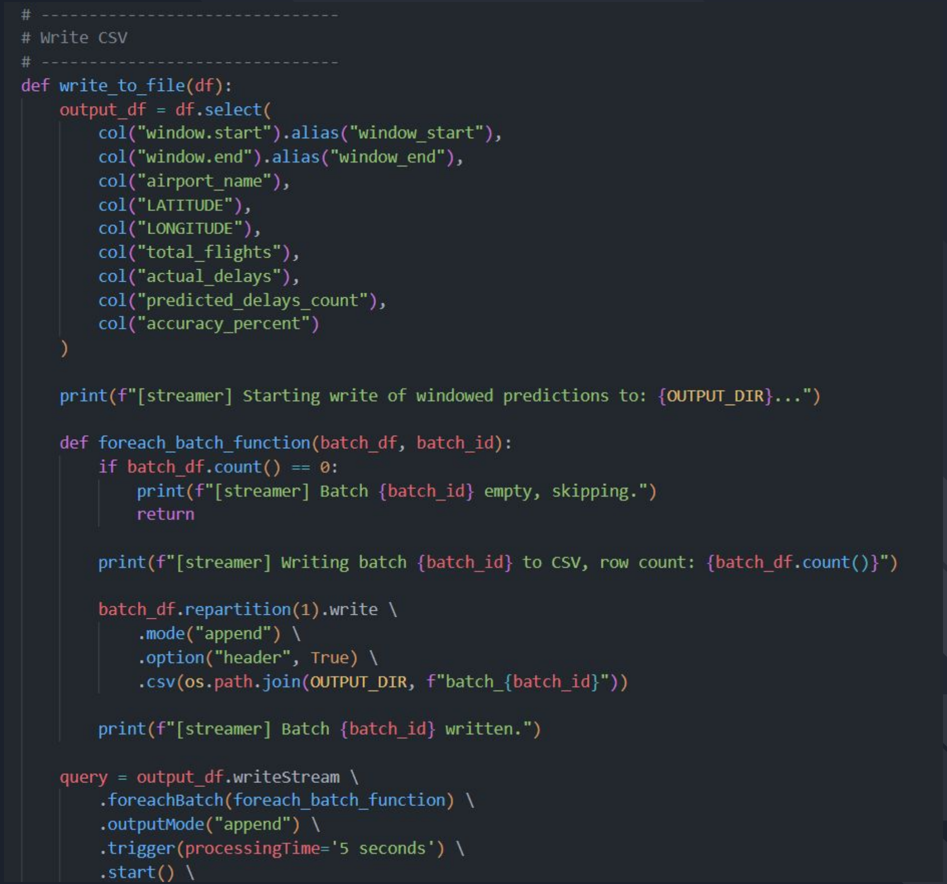
```python
# ------------------------------
# Aggregate by time window
# ------------------------------
def aggregate_and_predict_window(df, args):
    window_duration_str = f"{args.window_duration} seconds"

    if args.window_type.lower() == "tumbling":
        window_col = window(col("event_timestamp"), window_duration_str)
    else:
        slide_duration = args.slide_duration or args.window_duration
        slide_duration_str = f"{slide_duration} seconds"
        window_col = window(col("event_timestamp"), window_duration_str, slide_duration_str)

    df_with_watermark = df.withWatermark("event_timestamp", "3 minutes")

    windowed_predictions = df_with_watermark.groupBy(
        window_col,
        col("DEPARTING_AIRPORT").alias("airport_name")
    ).agg(
        count(col(TARGET_COL)).alias("total_flights"),
        sum(col(TARGET_COL)).alias("actual_delays"),
        sum(col("predicted_delay")).alias("predicted_delays_count"),

        sum(when(col(TARGET_COL).cast("int") == col("predicted_delay").cast("int"), 1).otherwise(0)).alias("correct_predictions"),

        max(col("LATITUDE")).alias("LATITUDE"),
        max(col("LONGITUDE")).alias("LONGITUDE")
    ).withColumn(
        "accuracy_percent",
        round(col("correct_predictions") / col("total_flights") * 100, 2)
    )

    return windowed_predictions
```

Spark streaming

- Upis CSV fajlova na OUTPUT_DIR putanji



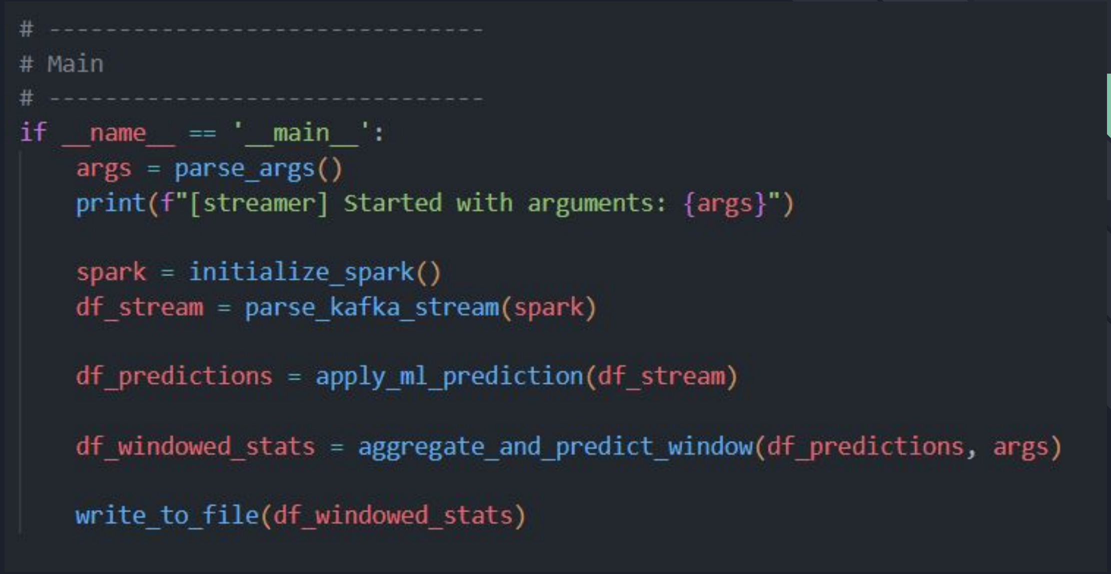
```
output\predictions
  batch_4
  batch_5
  batch_6
  batch_7
  batch_8
  batch_9
  batch_10
  batch_11
  batch_12
  batch_13
  batch_14
  batch_15
  batch_16
  batch_17
```



```python
# ------------------------------
# Write CSV
# ------------------------------
def write_to_file(df):
    output_df = df.select(
        col("window.start").alias("window_start"),
        col("window.end").alias("window_end"),
        col("airport_name"),
        col("LATITUDE"),
        col("LONGITUDE"),
        col("total_flights"),
        col("actual_delays"),
        col("predicted_delays_count"),
        col("accuracy_percent")
    )

    print(f"[streamer] Starting write of windowed predictions to: {OUTPUT_DIR}...")

    def foreach_batch_function(batch_df, batch_id):
        if batch_df.count() == 0:
            print(f"[streamer] Batch {batch_id} empty, skipping.")
            return

        print(f"[streamer] Writing batch {batch_id} to CSV, row count: {batch_df.count()}")

        batch_df.repartition(1).write \
            .mode("append") \
            .option("header", True) \
            .csv(os.path.join(OUTPUT_DIR, f"batch_{batch_id}"))

        print(f"[streamer] Batch {batch_id} written.")

    query = output_df.writeStream \
        .foreachBatch(foreach_batch_function) \
        .outputMode("append") \
        .trigger(processingTime='5 seconds') \
        .start() \
```

Spark streaming

- Main



```
# ------------------------------
# Main
# ------------------------------
if __name__ == '__main__':
    args = parse_args()
    print(f"[streamer] Started with arguments: {args}")

    spark = initialize_spark()
    df_stream = parse_kafka_stream(spark)

    df_predictions = apply_ml_prediction(df_stream)

    df_windowed_stats = aggregate_and_predict_window(df_predictions, args)

    write_to_file(df_windowed_stats)
```

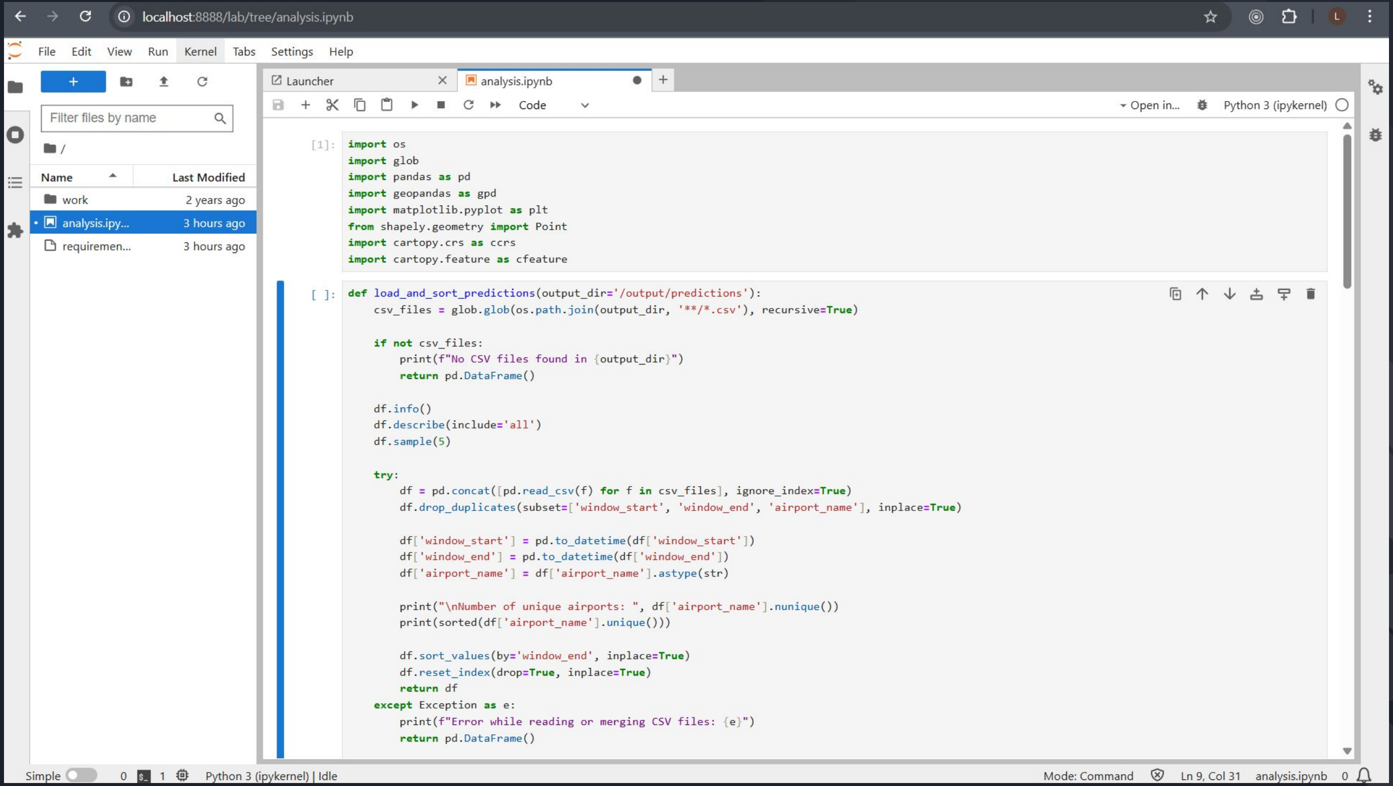
Vizualizacija rezultata - Jupyter Notebook

- Pokreće Jupyter Notebook na localhost:8888
- U notebook-u se koristi `visualization/analysis.ipynb` za:
  - učitavanje CSV batch-ova
  - vremensku analizu (plot_time_analysis)
  - geografske mape (plot_geographical_analysis)

- Pokretanje:
  - pozicioniranje u folder gde se nalazi `docker-compose-visualize.yml`
  - `docker-compose -f docker-compose-visualize.yml up --build`

Vizualizacija rezultata - Jupyter Notebook

- http://localhost:8888/
- Run all cells

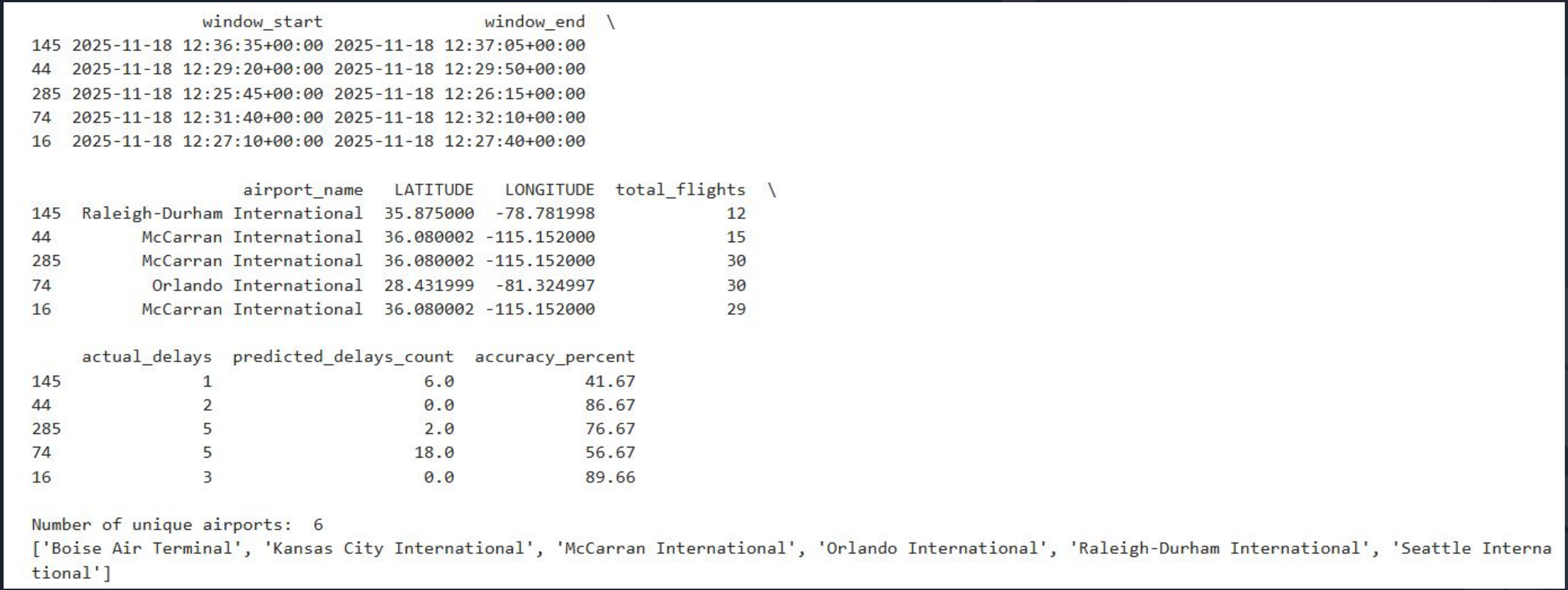

localhost:8888/lab/tree/analysis.ipynb
File Edit View Run Kernel Tabs Settings Help
Filter files by name
Name
Last Modified
work 2 years ago
analysis.ipy... 3 hours ago
requiremen... 3 hours ago
Launcher
analysis.ipynb
Code
Open in...
Python 3 (ipykernel)
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 9, Col 31
analysis.ipynb

```
[1]: import os
     import glob
     import pandas as pd
     import geopandas as gpd
     import matplotlib.pyplot as plt
     from shapely.geometry import Point
     import cartopy.crs as ccrs
     import cartopy.feature as cfeature
```

```
[ ]: def load_and_sort_predictions(output_dir='/output/predictions'):
         csv_files = glob.glob(os.path.join(output_dir, '**/*.csv'), recursive=True)

         if not csv_files:
             print(f"No CSV files found in {output_dir}")
             return pd.DataFrame()

         df.info()
         df.describe(include='all')
         df.sample(5)

         try:
             df = pd.concat([pd.read_csv(f) for f in csv_files], ignore_index=True)
             df.drop_duplicates(subset=['window_start', 'window_end', 'airport_name'], inplace=True)

             df['window_start'] = pd.to_datetime(df['window_start'])
             df['window_end'] = pd.to_datetime(df['window_end'])
             df['airport_name'] = df['airport_name'].astype(str)

             print("\nNumber of unique airports: ", df['airport_name'].nunique())
             print(sorted(df['airport_name'].unique()))

             df.sort_values(by='window_end', inplace=True)
             df.reset_index(drop=True, inplace=True)
             return df
         except Exception as e:
             print(f"Error while reading or merging CSV files: {e}")
             return pd.DataFrame()
```

Vizualizacija rezultata - Jupyter Notebook

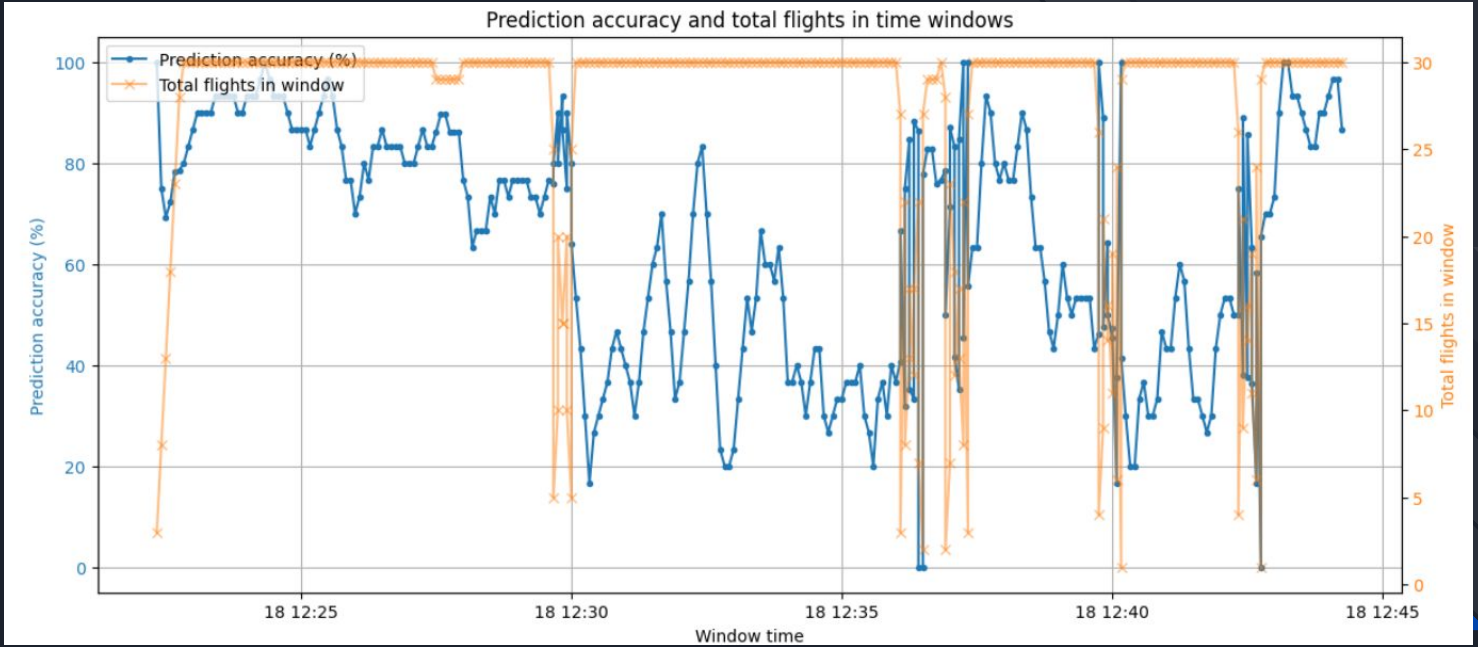
- Pregled DataFrame-a učitanog iz CSV fajla



```
                 window_start                window_end  \
145 2025-11-18 12:36:35+00:00 2025-11-18 12:37:05+00:00
44  2025-11-18 12:29:20+00:00 2025-11-18 12:29:50+00:00
285 2025-11-18 12:25:45+00:00 2025-11-18 12:26:15+00:00
74  2025-11-18 12:31:40+00:00 2025-11-18 12:32:10+00:00
16  2025-11-18 12:27:10+00:00 2025-11-18 12:27:40+00:00

                     airport_name   LATITUDE   LONGITUDE  total_flights  \
145  Raleigh-Durham International  35.875000  -78.781998             12
44         McCarran International  36.080002 -115.152000             15
285        McCarran International  36.080002 -115.152000             30
74          Orlando International  28.431999  -81.324997             30
16         McCarran International  36.080002 -115.152000             29

     actual_delays  predicted_delays_count  accuracy_percent
145              1                     6.0             41.67
44               2                     0.0             86.67
285              5                     2.0             76.67
74               5                    18.0             56.67
16               3                     0.0             89.66

Number of unique airports:  6
['Boise Air Terminal', 'Kansas City International', 'McCarran International', 'Orlando International', 'Raleigh-Durham International', 'Seattle Interna
tional']
```

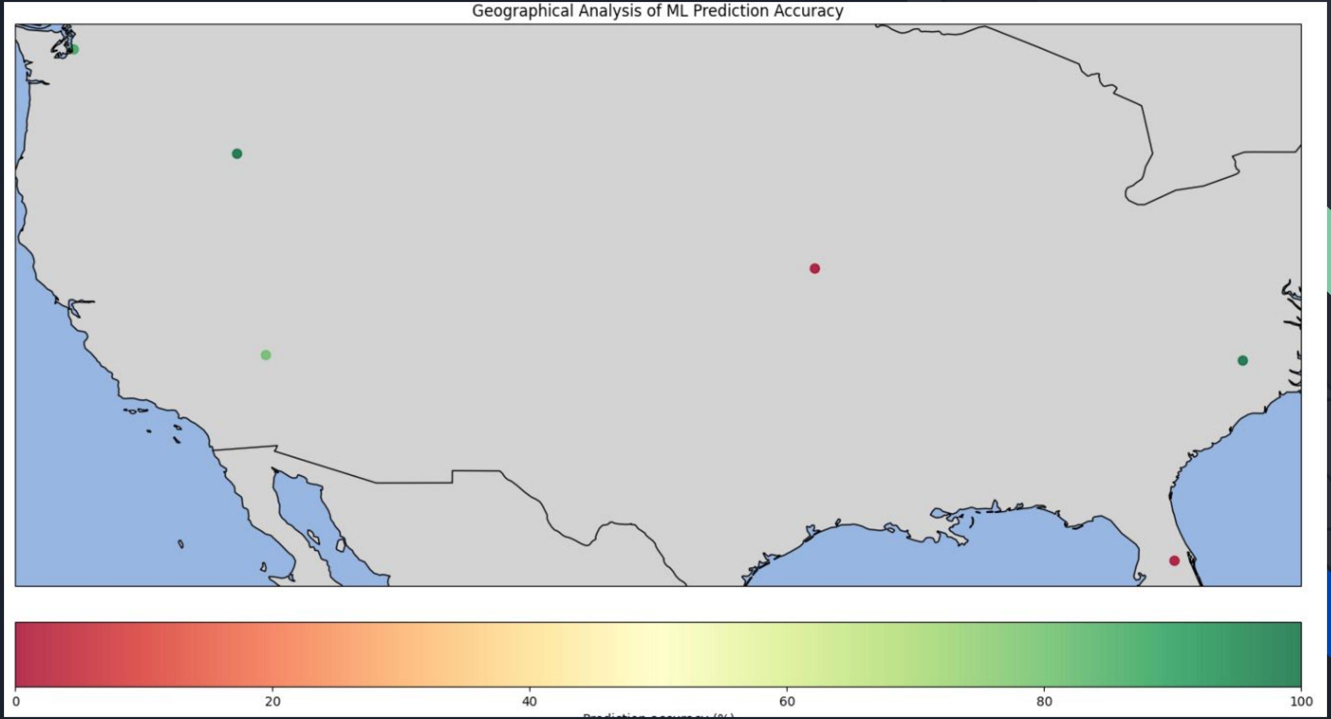
Vizualizacija rezultata - Jupyter Notebook

- Tačnost predviđanja i ukupan broj letova u vremenskim prozorima



Vizualizacija rezultata - Jupyter Notebook

- Geografska analiza tačnosti predviđanja mašinskog učenja



HVALA NA PAŽNJI!