Here is a short entry for each separate coding project I worked on in ITAI 4370.

---

1. Digital transmission simulation (Module 1 lab)

- **Problem statement**
  Simulate a simple digital telecom transmission chain to see how bits are generated, modulated, corrupted by noise, and recovered at the receiver.

- **Methods and tools used**
  Python, NumPy, and Matplotlib to generate random bits, map them to symbols, add noise, and compare original vs recovered bit streams.

- **Code snippet / repo**
  ```python
  bits = np.random.randint(0, 2, size=1000)
  symbols = 2*bits - 1                  # BPSK mapping
  noise = np.random.normal(0, 0.5, len(symbols))
  rx = symbols + noise
  bits_hat = (rx > 0).astype(int)
  ```

---

2. RF propagation & FSPL for WiFi bands (Lab 2, Module 2)

- **Problem statement**
  Quantify how free space path loss grows with distance for 2.4 GHz vs 5 GHz WiFi to understand wireless coverage tradeoffs.

- **Methods and tools used**
  Python, NumPy, Matplotlib; applied the FSPL formula in dB over a range of distances and plotted both frequency bands.

- **Code snippet / repo**
  ```python
  f1, f2 = 2400, 5000           # MHz
  d = np.linspace(0.1, 20, 100) # km
  fspl1 = 20*np.log10(d) + 20*np.log10(f1) + 32.44
  fspl2 = 20*np.log10(d) + 20*np.log10(f2) + 32.44
  plt.plot(d, fspl1, label="2.4 GHz")
  plt.plot(d, fspl2, label="5 GHz")
  ```

---

3. Network traffic prediction (Module 3 lab 1)

- **Problem statement**

  Predict near-term network traffic volume from historical usage patterns to support basic capacity planning.

- **Methods and tools used**
  Python, pandas, scikit-learn regression models; engineered features like hour-of-day and day-of-week and evaluated prediction error on a held-out set.

- **Code snippet / repo**
  ```python
  X = df[["hour", "dow", "prev_hour"]]
  y = df["traffic_mbps"]
  model = RandomForestRegressor(n_estimators=200, random_state=42)
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  ```

---

4. Network anomaly detection (Module 3 lab 2)

- **Problem statement**
  Detect unusual network traffic patterns (potential attacks or faults) from flow-level features.

- **Methods and tools used**
  Python, pandas, scikit-learn; trained an unsupervised anomaly detector (e.g., Isolation Forest) on mostly normal traffic and flagged outliers as anomalies.

- **Code snippet / repo**
  ```python
  from sklearn.ensemble import IsolationForest

  iso = IsolationForest(contamination=0.01, random_state=42)
  iso.fit(X_train)
  anomaly_scores = iso.decision_function(X_test)
  anomalies = iso.predict(X_test) == -1
  ```

---

5. Customer segmentation (Module 3 lab 3)

- **Problem statement**
  Group telecom customers into segments based on usage and behavior for targeted services or marketing.

- **Methods and tools used**
  Python, pandas, scikit-learn K-means clustering; normalized numeric features and interpreted cluster centroids to name segments.

- **Code snippet / repo**
  ```python
  from sklearn.preprocessing import StandardScaler
  from sklearn.cluster import KMeans

  X_scaled = StandardScaler().fit_transform(X)
  kmeans = KMeans(n_clusters=4, random_state=42)
  clusters = kmeans.fit_predict(X_scaled)
  df["segment"] = clusters
  ```

---

6. Network slicing with demand-based resource allocation (Module 4 lab)

- **Problem statement**
  Simulate a simple 5G network slicing scenario where resources are dynamically split between slices based on changing demand.

- **Methods and tools used**
  Python and NumPy; modeled total capacity and slice demands, then computed per-slice allocation each time step and tracked utilization.

- **Code snippet / repo**
  ```python
  total_bw = 100.0
  for t in range(T):
      d = np.array([d_slice1[t], d_slice2[t], d_slice3[t]])
      share = d / (d.sum() + 1e-8)
      alloc = total_bw * share
      usage_log.append(alloc)
  ```

---

7. ORAN traffic prediction with linear regression (Module 5 lab)

- **Problem statement**
  Show how a simple ML model can forecast ORAN traffic as a basic building block for AI-assisted RAN optimization.

- **Methods and tools used**
  Python, pandas, scikit-learn LinearRegression; used time-based features to fit a quick baseline model.

- **Code snippet / repo**
  ```python
  from sklearn.linear_model import LinearRegression

  X = df[["hour", "dow"]]
  ```

```
  y = df["traffic_gbps"]
  lr = LinearRegression()
  lr.fit(X_train, y_train)
  y_pred = lr.predict(X_test)
  ```

---

8. Multi-model traffic forecasting: ARIMA, linear regression, LSTM (Lab 4, Module 6)

- **Problem statement**
  Compare classical statistical and ML models for forecasting hourly network traffic on a real-world dataset.

- **Methods and tools used**
  Python, pandas, statsmodels ARIMA, scikit-learn LinearRegression, and Keras LSTM; performed train/test split, feature engineering, and evaluated each model with MAE/MSE/R².
  (Code in `ITAI_4370_Lab_04_Oren_Moreno.ipynb`.)

- **Code snippet / repo**
  ```python
  # ARIMA fit
  arima = ARIMA(train["traffic"], order=(2,1,2)).fit()
  arima_forecast = arima.forecast(steps=len(test))

  # Simple LSTM skeleton
  model = Sequential([
      LSTM(32, input_shape=(seq_len, n_features)),
      Dense(1)
  ])
  model.compile(optimizer="adam", loss="mse")
  ```

---

9. Edge AI model compression for IoT sensors (Lab 5, Module 7)

- **Problem statement**
  Take a cloud-scale NN for IoT sensor classification and compress it so it can run efficiently on constrained edge devices.

- **Methods and tools used**
  TensorFlow/Keras for the baseline model, pruning and quantization, and TensorFlow Lite conversion; simulated latency and accuracy for edge vs cloud.
  (Code in `ITAI_4370_Lab_05_Oren_Moreno_2025Fall.ipynb`.)

- **Code snippet / repo**
  ```python
  ```

```
  base_model = keras.Sequential([
      layers.Dense(64, activation="relu", input_shape=(n_features,)),
      layers.Dense(32, activation="relu"),
      layers.Dense(3, activation="softmax"),
  ])

  converter = tf.lite.TFLiteConverter.from_keras_model(base_model)
  converter.optimizations = [tf.lite.Optimize.DEFAULT]
  tflite_model = converter.convert()
  ```
```

---

### 10. ML-based IDS on NSL-KDD (Module 8 lab)

- **Problem statement**
  Build an intrusion detection system that can classify network connections as normal or specific attack types using NSL-KDD.

- **Methods and tools used**
  Python, pandas, scikit-learn RandomForest and SVM classifiers, plus Isolation Forest and One-Class SVM for unsupervised detection; used SHAP for feature importance explanations.

- **Code snippet / repo**
  ```python
  rf = RandomForestClassifier(n_estimators=300, random_state=42)
  rf.fit(X_train, y_train)
  y_pred = rf.predict(X_test)

  explainer = shap.TreeExplainer(rf)
  shap_values = explainer.shap_values(X_test_sample)
  ```

---

11. Energy forecasting & base-station sleep scheduling (Module 9 lab)

- **Problem statement**
  Forecast telecom energy consumption and design a base-station sleep strategy that reduces cost and emissions while keeping QoS acceptable.

- **Methods and tools used**
  Python, Prophet and LSTM models for energy/time-series prediction; evaluated different sleep windows and computed energy and carbon savings.

- **Code snippet / repo**
  ```python
  # Prophet
  m = Prophet()
```
```

```
  m.fit(df_prophet)
  forecast = m.predict(future)

  # Simple sleep rule
  df["sleep"] = (df["pred_traffic"] < threshold).astype(int)
  saved_kwh = (df["sleep"] * base_station_power * dt).sum()
  ```

---

12. Digital twin of a 3-router network with SimPy (Midterm Q1, Module 10)

- **Problem statement**
  Build a live digital twin of a small network (three routers in series) to study throughput, latency, and queue growth as traffic conditions change.

- **Methods and tools used**
  SimPy for discrete-event simulation, NumPy and statistics for metrics, Matplotlib for time series plots of queue lengths and latency.
  (Code in `ITAI_4370_Midterm_Oren_Moreno_2025Fall.ipynb`.)

- **Code snippet / repo**
  ```python
  def router(env, in_q, out_q, service_time):
      while True:
          pkt = yield in_q.get()
          yield env.timeout(service_time)
          yield out_q.put(pkt)

  env = simpy.Environment()
  r1_q, r2_q, r3_q = simpy.Store(env), simpy.Store(env), simpy.Store(env)
  env.process(router(env, r1_q, r2_q, 0.5))
  ```

---

13. Agent-based network modeling with Mesa (Midterm Q2)

- **Problem statement**
  Use an agent-based model to simulate packets flowing through a network graph and observe congestion hotspots and routing behavior.

- **Methods and tools used**
  Mesa for ABM structure, NetworkX for graph topology, and Matplotlib for visualizing node load and packet paths.
  (Code in `ITAI_4370_Midterm_Oren_Moreno_2025Fall.ipynb`.)

- **Code snippet / repo**
  ```python
  class RouterAgent(Agent):
```

```
      def __init__(self, unique_id, model):
          super().__init__(unique_id, model)
          self.queue = []

      def step(self):
          if self.queue:
              pkt = self.queue.pop(0)
              next_hop = random.choice(self.model.graph.neighbors(self.unique_id))
              self.model.route_packet(pkt, next_hop)
```

---

14. Predictive optimization of routing with ML (Midterm Q3)

- **Problem statement**
  Learn a model of future traffic and use it to dynamically split load across two parallel paths in order to minimize total latency compared to a static policy.

- **Methods and tools used**
  Python, scikit-learn RandomForestRegressor for single-step traffic prediction; applied a latency cost function and compared predictive vs static routing policies.

- **Code snippet / repo**
  ```python
  model = RandomForestRegressor(n_estimators=300, random_state=42)
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)

  s = np.clip(alpha / (y_pred + 1e-3), 0.1, 0.9)   # split to path 1
  latency_ml = latency(path1, s*y) + latency(path2, (1-s)*y)
  ```

---

15. Federated learning simulation for 6G resource allocation (Module 11 lab)

- **Problem statement**
  Simulate a federated learning setup where multiple edge nodes collaboratively train a model for predictive resource allocation without sharing raw data.

- **Methods and tools used**
  Python and PyTorch; each client trained locally on its partitioned data, sent model updates to a central server for weighted averaging (FedAvg), and then evaluated global performance vs a centralized baseline.

- **Code snippet / repo**
  ```python
  def client_update(model, loader, epochs=1):
      opt = torch.optim.SGD(model.parameters(), lr=0.01)
```

```
    for _ in range(epochs):
        for x, y in loader:
            opt.zero_grad()
            loss = criterion(model(x), y)
            loss.backward()
            opt.step()
    return model.state_dict()

# Server side aggregation
global_state = average_states(client_states, weights)
```