VIVA — Code-step Explanations (Concise)

Notebook code cells (only code cells explained):

1. Imports

- import pandas as pd; import numpy as np; from sklearn.preprocessing import LabelEncoder
- Explanation: load libraries for data handling and preprocessing.

2. Load data

- df = pd.read_csv("Student_performance_data _.csv")
- Explanation: read the dataset into a DataFrame named `df`.

3. Inspect rows

- df.head(5); df.tail(5)
- Explanation: display top and bottom rows to check structure and sample values.

4. Shape and columns

- print('rows: ', df.shape[0]); print('columns: ', df.shape[1]); df.columns
- Explanation: confirm dataset size and column names.

5. Missing values check

- df.isnull().sum()
- Explanation: count nulls per column to plan imputation.

6. Drop column

- df = df.drop('age', axis=1)
- Explanation: remove `age` column from dataset.

7. Fill missing values

- df = df.fillna(df.mode)
- Explanation: fill missing entries using mode (most frequent values).

8. Fill specific object columns

- define fillnaObjectMode(cols) and call for ['Sports','Age','Gender']
- Explanation: fill nulls in categorical columns with each column's mode.

9. Data types and conversions

- df.dtypes; df.StudyTimeWeekly = df.StudyTimeWeekly.astype(np.int64); df.GPA = df.GPA.astype(np.int64); df.GradeClass = df.GradeClass.astype(np.int64)
- Explanation: inspect types and cast selected columns to integer.

10. Drop Extracurricular

- df.drop('Extracurricular', axis=1, inplace=True)
- Explanation: remove the `Extracurricular` column.

11. Split features and target (generic)

- x = df.iloc[:, 0:-1]; y = df.iloc[:,-1]
- Explanation: assign feature matrix `x` and target `y`.

12. Encode categorical columns

- cat_columns = x.select_dtypes(['object']).columns; x[cat_columns] = x[cat_columns].apply(lambda x: pd.factorize(x)[0])
- Explanation: convert string categories to integer codes.

13. Prepare classifier data (not the final regression)

- X = df.drop('Gender', axis=1); y = df['Gender']
- Explanation: set `X` and `y` to train a Gender classifier in the notebook.

14. Train/test split

- train_test_split(X,y, test_size=0.3, shuffle=False)
- Explanation: split data into training and test sets (30% test). Note: shuffle=False preserves order.

15. Train classifier (SVC)

- model_svc = SVC(); model_svc.fit(train_X, train_y)
- Explanation: fit an SVC classifier to training data.

16. Predict and evaluate (classifier)

- model_pred_svc = model_svc.predict(test_X); accuracy_score(test_y, model_pred_svc)
- Explanation: compute predictions and classification accuracy on test set.

17. Save and load classifier

- pickle.dump(model_svc, open('model_svc.pkl','wb')); model_svc = pickle.load(open('model_svc.pkl','rb'))
- Explanation: serialize and deserialize the trained classifier using pickle.

app.py — step-by-step explanation (concise):

1. Imports and setup

- import pickle, os, numpy as np; from flask import Flask, render_template, request
- Explanation: load pickle for model I/O, os for file paths, numpy for array operations and Flask for the web app.

2. Determine app root and model path

- APP_ROOT = os.path.dirname(os.path.abspath(**file**)); MODEL_PATH = os.path.join(APP_ROOT, "model.pkl")
- Explanation: compute absolute paths so the app reliably finds `model.pkl`.

3. Create Flask app

- app = Flask(**name**, template_folder="templates")
- Explanation: initialize Flask and set the templates folder.

4. Load saved model and features

- with open(MODEL_PATH,'rb') as f: saved = pickle.load(f)
- model = saved['model']; FEATURE_NAMES = saved['features']
- Explanation: load a dict containing the trained model and the ordered feature names.

5. Boolean features list

- BOOL_FEATURES = [f for f in FEATURE_NAMES if f.lower() in ('tutoring','extracurricular','sports','music','volunteering')]
- Explanation: mark known boolean features so checkboxes are handled correctly.

6. Feature importances

- if hasattr(model,'feature_importances_'): compute and sort feature importances
- Explanation: prepare a sorted list of (feature,importance) for display when available.

7. Root route

- @app.route('/') def home(): return render_template('index.html')
- Explanation: serve the input form page.

8. Predict route (core flow)

- @app.route('/predict', methods=['POST'])
  - Loop over FEATURE_NAMES and for each name:
    - if name in BOOL_FEATURES: raw = request.form.get(name); val = 1.0 if raw in ('on','1','true','True') else 0.0
    - else: raw = request.form.get(name); try: val = float(raw) except: val = 0.0
    - append val to vals list
  - user_input = np.array([vals])

    - prediction = float(model.predict(user_input)[0])
    - if hasattr(model,'estimators_'): tree_preds = np.array([est.predict(user_input)[0] for est in model.estimators_]); uncertainty = float(tree_preds.std())
    - return render_template('results.html', prediction=round(prediction,3), uncertainty=uncertainty, importances=FEATURE_IMPORTANCES)
- Explanation: read form fields in the saved order, convert values to numeric (checkboxes -> 0/1), build a 2D array for sklearn, predict, compute simple ensemble uncertainty, and render results.

9. Error handling

- try/except in predict route: on exception render `error.html` with the error string
- Explanation: show a friendly error page instead of crashing.

10. App runner

- if **name** == "**main**": app.run(debug=True)
- Explanation: start the Flask development server when running the script directly.

Explanation of .pkl files (concise):

- model.pkl
    - Contains a pickled dictionary: {'model': trained_regressor, 'features': [ordered feature names]}
    - Explanation: stores the trained model plus the exact feature order required by the app.
- model_svc.pkl
    - Contains a pickled SVC classifier trained in the notebook to predict `Gender`.
    - Explanation: an earlier experiment; not used by the GPA web app.

Explanation of template and static files (concise):

- templates/index.html
    - Contains the input form. Field `name` attributes match `FEATURE_NAMES` so the server reads values in the correct order. Checkboxes represent boolean features.
- templates/results.html
    - Displays `{{ prediction }}`, optional `{{ uncertainty }}` and the top feature importances passed from `app.py`.
- static/styles.css
    - Styling for layout and presentation. No logic; purely visual.

Notes (single-line):

- The saved `features` list guarantees consistent input ordering between training and prediction.
- Checkbox values are converted server-side to 0/1 to ensure numeric input to the model.
- Ensemble uncertainty shown is the standard deviation of tree predictions (a quick spread measure, not a calibrated interval).