Project Explanation (SalaryCast)

🡺 app.py

import streamlit as st

import numpy as np

import pickle         # Used for loading the pre-trained model from disk

import random         # Used for generating random positions and delays for the money animation

# Load the model 👇

"""

pickle.load(...): Loads a previously saved model.

open(..., 'rb'): Opens the file in read-binary mode.

'r' = read

'b' = binary mode (used for non-text files like models)

We use 'rb' because machine learning models are saved in binary format.

"""

==================================================================================

model = pickle.load(open(r'D:\nit\_prac\11th  june\linear\_regression\_model.pk1', 'rb'))

# Custom CSS + HTML Title

"""

st.markdown(...)  --->  Renders text using Markdown syntax (bold, italics, headers, links, etc.).

⚠️ unsafe\_allow\_html=True

Enables rendering of raw HTML tags inside the string.

Useful when you want custom styles or HTML elements that Markdown doesn't support (like <div>, <span>, <img>, or inline CSS).

🚫 Why it’s called “unsafe”?

Allowing raw HTML can be unsafe if the HTML content comes from untrusted users (risk of script injection or XSS attacks). Streamlit disables it by default for safety.

"""

==================================================================================

/\* Money animation \*/

@keyframes floatUp

            {

            0%   { transform: translateY(0); opacity: 1; }

            100% { transform: translateY(-120vh); opacity: 0; }

            }

        .money {

            position: fixed;

            font-size: 30px;

            animation: floatUp 3s ease-in-out;

            z-index: 9999;

            pointer-events: none;

        }

**@keyframes floatUp**

This defines an animation named floatUp.

**It controls how the element moves and fades:**

* 0% → start of animation:
  + transform: translateY(0) → start at current Y position (bottom of screen).
  + opacity: 1 → fully visible.
* 100% → end of animation:
  + transform: translateY(-120vh) → move up 120% of viewport height (i.e., fly far above screen).
  + opacity: 0 → completely transparent (fades out).

So, the element will **float upward and fade away**.

**🎯 .money class styling**

This styles the animated emoji elements (💰, 💵, 💸):

| **Property** | **Meaning** |
| --- | --- |
| position: fixed | Sticks the emoji in a fixed spot on the screen (not affected by scroll). |
| font-size: 30px | Size of the emoji. |
| animation: floatUp 3s ease-in-out | Applies the floatUp animation for **3 seconds**, with smooth acceleration and deceleration. |
| z-index: 9999 | Puts the emoji **on top of everything else** on the page. |
| pointer-events: none | Makes the emoji **non-clickable** and ignores mouse interaction. |

==================================================================================

# Input

* Shows a numeric input box.
* Accepts values from 0 to 50 in 0.5 steps.
* Default value is 1.0.

yr\_exp = st.number\_input("📅 Enter years of experience:", min\_value=0.0, max\_value=50.0, value=1.0, step=0.5)

==================================================================================

# Predict Button

- Creates a button labelled "Predict Salary".

- Runs the block of code only when clicked.

if st.button("🔍 Predict Salary"):

    exp\_input = np.array([[yr\_exp]]) # Converts input into a 2D array (shape required by the model).

    pred = model.predict(exp\_input)[0]

- Predicts the salary using the ML model.

- [0] extracts the number from the returned array.

formatted\_salary = "${:,.2f}".format(pred)

* Formats the predicted salary as currency (e.g., $52,000.00).

================================================================================

# Display output – result box

* Uses **f-string (f'''...''')** to insert dynamic values like yr\_exp and formatted\_salary.
* <b> …. </b> bold tag

st.markdown(f'''

<div class="result-box">✅ Predicted salary for <b>{yr\_exp}</b> year(s) of experience is:<br><b>{formatted\_salary}</b> </div>

''', unsafe\_allow\_html=True)

================================================================================

# money rain logic

**money\_html = ""**

* Initializes an **empty string**.
* This string will be filled with HTML for the flying money emojis.

**2. for \_ in range(3):**

* Outer loop: Runs **3 times** to create **3 bursts** of emoji.

**3. for \_ in range(10):**

* Inner loop: Runs **10 times per burst**.
* So: 3 × 10 = **30 animated emoji elements** in total.

**4. left = random.randint(0, 95)**

* Chooses a random **horizontal position** (from 0% to 95% of screen width).
* This makes emojis fall from different positions across the top.

**5. emoji = random.choice(['💸', '💵', '💰'])**

* Randomly picks one of the **money emojis** to use.

**6. delay = random.uniform(0, 2)**

* Chooses a random **delay (0 to 2 seconds)**.
* This staggers the animations, so not all emojis fall at the same time.

**7. money\_html += f'''...'''**

* Appends the generated HTML to the money\_html string.
* The HTML is:

html

Copy code

<div class="money" style="left:{left}%; top:100%; animation-delay:{delay}s;">{emoji}</div>

**Let's break this inner HTML part:**

html

Copy code

<div class="money"

style="left:{left}%;

top:100%;

animation-delay:{delay}s;">

{emoji}

</div>

| **Attribute** | **What it does** |
| --- | --- |
| class="money" | Uses the CSS class you defined earlier (makes the emoji float up and fade) |
| left:{left}% | Puts the emoji at a random spot from left (0% to 95%) |
| top:100% | Starts from the bottom of the screen |
| animation-delay:{delay}s | Starts the floating animation after some delay (staggered launch) |
| {emoji} | The actual emoji shown (💸, 💵, or 💰) |

**8. st.markdown(money\_html, unsafe\_allow\_html=True)**

* Displays all the 30 <div> emoji elements in Streamlit.
* Because unsafe\_allow\_html=True, the **floating money animation** is rendered properly using the custom CSS and HTML

money\_html = ""

for \_ in range(3): # repeat 3 times

for \_ in range(10): # 10 emojis per burst

left = random.randint(0, 95) # horizontal position

emoji = random.choice(['💸', '💵', '💰'])

delay = random.uniform(0, 2) # delay to stagger

money\_html += f'''

<div class="money" style="left:{left}%; top:100%; animation-delay:{delay}s;">{emoji}</div>

'''

# Inject animated money HTML

st.markdown(money\_html, unsafe\_allow\_html=True)

================================================================================

🡺 Salary\_prac.py

**🔍 Sum of Squares (Important)**

**1. SSR: Sum of Squares due to Regression**

y\_mean = np.mean(y)

SSR = np.sum((y\_pred - y\_mean) \*\* 2)

* Measures **explained variance**: how much the model explains the variation in the data.
* Formula: SSR = ∑(ŷ - ȳ)²  
  Where:
  + ŷ = predicted values
  + ȳ = mean of actual y

**2. SSE: Sum of Squared Errors**

y = y[0:6]

SSE = np.sum((y - y\_pred) \*\* 2)

* Measures **unexplained variance**: difference between actual and predicted.
* Formula: SSE = ∑(y - ŷ)²

⚠️ Note: You trimmed y = y[0:6] — this affects correctness if y\_pred has more than 6 values. This should match y\_pred’s shape.

**3. SST: Total Sum of Squares**

mean\_Total = np.mean(dataset.values)

SST = np.sum((dataset.values - mean\_Total) \*\* 2)

* Measures **total variance** in data.
* Formula: SST = ∑(y - ȳ)² (But you're using the whole dataset, which is technically incorrect for Salary vs. Experience analysis).
* Recommended version:

SST = np.sum((y\_test - y\_mean) \*\* 2)

**🧮 R² Score (Coefficient of Determination)**

r\_square = 1 - (SSR / SST)

* R² = how much variance the model explains.
* R² = 1 → perfect model; R² = 0 → model explains nothing.

**💾 Save Trained Model**

python

Copy code

import pickle

filename = 'linear\_regression\_model.pk1'

with open(filename, 'wb') as file:

pickle.dump(regressor, file)

* Saves model to disk using pickle.
* 'wb': write binary.

import os

print(os.getcwd())

* Prints the **current working directory** (where the model is saved).

**Summary of SST, SSR, SSE:**

| **Metric** | **Formula** | **Meaning** |
| --- | --- | --- |
| **SST** | ∑(y - ȳ)² | Total variation in the data |
| **SSR** | ∑(ŷ - ȳ)² | Explained by the model |
| **SSE** | ∑(y - ŷ)² | Error (not explained by model) |
| **R²** | 1 - SSE/SST | % of variance explained by model |

🡺 Outputs

Plotting:



Slope: [9312.57512673]

Intercept: 26780.099150628186

y\_12: [138531.00067138]

compare predicted and actual salaries from test set:

Actual Predicted

2 37731 40748.961841

28 122391 122699.622956

13 57081 64961.657170

10 63218 63099.142145

26 116969 115249.562855

24 109431 107799.502753

Bias : 0.9411949620562126

variance : 0.988169515729126

**Mean:**

YearsExperience 5.313333

Salary 76003.000000

dtype: float64

**Median**

YearsExperience 4.7

Salary 65237.0

dtype: float64

**Variance of all columns:**

YearsExperience 8.053609e+00

Salary 7.515510e+08

**Variance of Salary column:** 751550960.4137931

**Standard deviation:**

YearsExperience 2.837888

Salary 27414.429785

dtype: float64

**Coefficient of variation:**

[0.5251297 0.35463929]

**Correlation matrix:**

YearsExperience Salary

YearsExperience 1.000000 0.978242

Salary 0.978242 1.000000

**Skeweness:**

YearsExperience 0.37956

Salary 0.35412

dtype: float64

**Standard Error:**

YearsExperience 0.518125

Salary 5005.167198

dtype: float64

**Standard Error individual:**5005.167198052405

**Z-Score of entire dataframe:**

[[-1.51005294 -1.36011263]

[-1.43837321 -1.10552744]

[-1.36669348 -1.419919 ]

[-1.18749416 -1.20495739]

[-1.11581443 -1.33978143]

[-0.86493538 -0.71830716]

[-0.82909552 -0.58815781]

[-0.75741579 -0.79981746]

[-0.75741579 -0.42881019]

[-0.57821647 -0.69801306]

[-0.50653674 -0.47433279]

[-0.47069688 -0.74976858]

[-0.47069688 -0.70662043]

[-0.43485702 -0.70201994]

[-0.29149756 -0.55250402]

[-0.1481381 -0.29921736]

[-0.07645838 -0.37004264]

[-0.00477865 0.26285865]

[ 0.21026054 0.19885989]

[ 0.2461004 0.66547573]

[ 0.53281931 0.58377993]

[ 0.6403389 0.82623317]

[ 0.92705781 0.93861127]

[ 1.03457741 1.40274136]

[ 1.21377673 1.24020308]

[ 1.32129632 1.09740238]

[ 1.50049564 1.51986835]

[ 1.5363355 1.3590738 ]

[ 1.78721455 1.72102849]

[ 1.85889428 1.70177321]]

**Degree of Freedom:** 28

**y\_mean :** 76003.0

**SSR:** 6263152884.284127

**SSE:** 15274062883.9432

**SST:** 108429703765.82735

**R\_Square:** 0.9422376648947554

model has been pickled and saved as linear\_regression\_model.pk1

D:\nit\_prac\11th June