Tarea de Investigación

- **Nombres:** Felipe Peralta y Samantha Suquilanda
- **Fecha:** Cuenca, 22 de octubre 2025

Parte 1: Estimación del Mercado Laboral Tech en Ecuador (Expandido)

A continuación, se presenta una tabla detallada con estimaciones de salarios, responsabilidades y tecnologías clave para roles de tecnología en Ecuador.

Nota:

- Los salarios son estimaciones para roles de nivel **Intermedio**

Profesión	Volumen de Ofertas (Demanda)	Salario Estimado (USD/hora)	Salario Estimado (USD/Anual)	Responsabilidades Clave	Frameworks / Tecnologías Comunes
Frontend Engineer	**Alto**	$5.50 - $10.50	$11,440 - $21,840	Implementar la interfaz de usuario (UI) y la experiencia de usuario (UX). Consumir APIs. Asegurar la interactividad y el *responsiveness* del sitio.	**React.js**, **Angular**, **Vue.js**, HTML5, CSS3/Sass.
Backend Engineer	**Alto**	$6.00 - $11.50	$12,480 - $23,920	Desarrollar la lógica del servidor, administrar bases de datos, crear APIs (REST/GraphQL) y manejar la autenticación y seguridad.	**Node.js** (Express), **Python** (Django, Flask), **Java** (Spring Boot), **.NET**, SQL (PostgreSQL, MySQL).

Profesión	Volumen de Ofertas (Demanda)	Salario Estimado (USD/hora)	Salario Estimado (USD/Anual)	Responsabilidades Clave	Frameworks / Tecnologías Comunes
Data Engineer	**Medio**	$6.50 - $13.00	$13,520 - $27,040	Diseñar, construir y mantener *pipelines* de datos (ETLs/ELTs). Gestionar la ingesta, almacenamiento y transformación de grandes volúmenes de datos.	**Apache Spark**, **Airflow**, SQL/NoSQL, Kafka, Databricks, (A veces **MLflow** o **Kubeflow** para orquestar pipelines).
ML Engineer	**Bajo**	$8.00 - $16.00	$16,640 - $33,280	Poner modelos de Machine Learning en producción (MLOps). Optimizar modelos para inferencia, crear APIs para su consumo y monitorear su rendimiento.	**Scikit-learn**, **TensorFlow**, **Keras** (o PyTorch), **MLflow**, **Kubeflow**, Docker, Kubernetes.
AI Engineer	**Muy Bajo**	$8.50 - $17.50+	$17,680 - $36,400+	Investigar, entrenar e implementar modelos complejos (Deep Learning, LLMs, Computer Vision, NLP). Es un rol más enfocado en R&D que el ML Engineer.	**TensorFlow**, **Keras**, **PyTorch**, Hugging Face Transformers, LangChain, (Scikit-learn para tareas base).

Observaciones

1. **Roles de Datos (Data/ML/AI):** Los frameworks como (**Scikit-learn**, **TensorFlow**, **Keras**, **MLflow**, **Kubeflow**) son el estándar de la industria.
    - **Scikit-learn:** Usado por casi todos para ML clásico.
    - **TensorFlow/Keras:** Dominantes en Deep Learning (junto con PyTorch).
    - **MLflow/Kubeflow:** Son herramientas de MLOps (Machine Learning Operations) para gestionar el ciclo de vida del modelo, siendo cruciales para los ML Engineers.
2. **Demanda en Ecuador:** La demanda local de AI/ML Engineers puros es baja y se concentra en *startups* de tecnología o centros de innovación de grandes empresas. Sin embargo, la demanda de *Data Engineers* está creciendo muy rápido.
3. **El Factor Remoto:** Los salarios para todos estos roles se disparan si el profesional trabaja de forma remota para empresas de EE. UU. o Europa, pudiendo fácilmente duplicar o triplicar las cifras anuales mostradas.

Importancia de Roles: Data Engineer y Feature Engineer

Es primordial conocer sobre los conceptos básicos del tema, antes de poder aplicarlo y resolver el Query.

1. **Ingeniería de Características (Feature Engineering / FE):** es el proceso de usar el conocimiento del dominio para transformar datos crudos en variables (características o features) que representan de mejor manera el problema subyacente.

- **El objetivo es simple:** facilitar el trabajo del modelo de Machine Learning. La calidad de las características que alimentas a un modelo tiene un impacto mucho mayor en el resultado final que el algoritmo específico que elijas.
- Esta se divide principalmente en:
    1. Creación de características: crear nuevas variables a partir de las existentes. Ejemplo: Del dato fecha_construccion: Puedes crear antiguedad_casa (Año actual - fecha_construccion).
    2. Transformación y limpieza: Los modelos matemáticos necesitan números limpios y en formatos específicos. Ejemplo: Codificación (Encoding): Convertir texto a números.
    3. Selección de características: Después de crear docenas o cientos de características, debes eliminar las que no aportan información (ruido) o las que son redundantes.

2. **Data Engineer:** Es el **habilitador** clave. No suele diseñar el modelo de similaridad, pero es responsable de construir y mantener el *pipeline* de datos que:
    - Genera millones de embeddings (que es costoso).
    - Los almacena y optimiza en bases de datos especializadas (como *Vector Databases*) para que el AI Engineer pueda hacer consultas de similaridad en milisegundos.

Conceptos Clave

Similaridad Coseno: mide el ángulo entre dos vectores.

- Si los vectores apuntan en la misma dirección, su similaridad es 1 (significan lo mismo).
- Si son perpendiculares (no tienen nada que ver), su similaridad es 0.
- Si apuntan en direcciones opuestas, es -1.

Ejercicio a Resolver

Query: comparar el significado de "dame el total de la factura IQ5430" con el significado de las funciones:

```
- Funcion_1: "find_invoice"

- Funcion_2: "find_user"

- Funcion_3: "create_invoice"
```

- **Análisis:** es un problema clásico de llamada a funciones (function calling) en sistemas de IA.
    - **Parte 1:** La medida que necesitamos no es una simple, sino una combinación de dos técnicas:

```
1. Embeddings (Vectores de Significado)

2. Similaridad Coseno (Cosine Similarity)
```

La medida de similaridad por sí sola no funciona con texto crudo. Primero debemos convertir el query y funciones a un formato numérico que entienda de significado (Transformación y limpieza).

- **Pasos Para Resolver**

1. **Vectorización (Crear Embeddings):** primero debemos "vectorizar" (crear embeddings) tanto para el query como las funciones.
    - Un embedding es un vector de números que representa el significado semántico del texto.
    - **Paso 1.1: Vectorizar el Query**

      a. Al query: "dame el total de la factura IQ5430"

      b. Lo pasamsos por un modelo de embeddings (ej. sentence-transformers en Python).

      Resultado: Vector_Query = [0.12, -0.45, 0.81, ...]

    - **Paso 1.2: Vectorizar las Funciones**

      Es importante no solo vectorizar el nombre (find_invoice) sino también una descripción clara de lo que hace la función.

```
  - Funcion_1: "find_invoice: Busca y devuelve los detalles de
una factura usando su ID."

  - Funcion_2: "find_user: Encuentra los datos de un usuario por
su nombre o ID."

  - Funcion_3: "create_invoice: Genera una nueva factura para un
cliente."
```

      Pasamos cada una por el modelo de embeddings, y tenemos:

```
  - Vector_Func1 = [0.10, -0.40, 0.79, ...] (Muy similar a
Vector_Query)

  - Vector_Func2 = [-0.50, 0.11, -0.23, ...] (Muy diferente)
```

```
   - Vector_Func3 = [0.05, -0.30, 0.65, ...] (Algo similar, pero
menos)
```

2. **Medición (Calcular Similaridad Coseno):** ahora que todos son vectores, se usa la Similaridad Coseno para comparar el Vector_Query contra cada Vector_Func. Y obtenemos:
    - score_1 = cosine_similarity(Vector_Query, Vector_Func1) / Resultado: 0.92 (muy alto)
    - score_2 = cosine_similarity(Vector_Query, Vector_Func2) / Resultado: 0.15 (muy bajo)
    - score_3 = cosine_similarity(Vector_Query, Vector_Func3) / Resultado: 0.61 (medio relacionado)
3. **Ranking (Obtener el Top 3)**

    Finalmente, ordenamos los resultados de mayor a menor:

    Top 3 de funciones más relacionadas:

```
1. find_invoice (Score: 0.92)

2. create_invoice (Score: 0.61)

3. find_user (Score: 0.15)
```

Conclusión

Con seguridad decimos que la función que se debe llamar es **find_invoice**