Skills Network

Hands-on Lab: Generative AI for Data Analysis and Mining

Estimated Effort: 30 mins

Introduction

One of the final tasks performed by a data engineer is to analyze the final data, draw insights from it, and employ data mining strategies to extract hidden patterns in the data distribution. In this lab, you will learn how to use generative AI for creating Python codes that can perform the required data analysis and data mining strategies.

Scenario

As a senior data engineer for a healthcare company, you are tasked to perform data analysis and data mining on patients' health records indicating whether or not the patient has been identified with a liver disease or not. Other teams have recorded and cleaned the data that is ready for analysis.

Objectives

In this lab, you will learn how to use generative AI to:

1. Perform exploratory data analysis on a given data set.

2. Perform data mining on the given data set and draw insights from the data.

About generative AI classroom lab

▶ Click here

Notes:

1. The prompts used in this lab are for your reference only. You can create your own prompts and generate responses using generative AI.
2. Since AI-generated outputs are dynamic, you may receive different responses even though you've used the same prompt from this lab.

Data set

For the purpose of this lab, we are making use of the [Indian Liver Patient Dataset](), publically available under the [Creative Commons Attribution 4.0 International (CC BY 4.0)]() license. You may refer to the data set web page for more details on the attributes.

The data set is available for use in this lab at the following URL:

```
URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMSkillsNetwork-AI0273EN-SkillsNetwork/labs/v1/m2/data/IL
```

Testing Interface

You will find a separate testing interface at the end of the lesson on the course page. Please keep that testing interface open as a separate lab on the side and follow the initial steps to be ready with the setup.

Exploratory Data Analysis

With exploratory data analysis, you can identify the type of data distribution and how different variables affect each other. In this task, you will use generative AI to create a code that does the following:

1. Create a histogram plot of the data based on the 'Age', 'Gender' and 'Selector' attributes.

2. Create a correlation heat map for the data set.

3. Select the top 5 features with the highest correlation value with the 'Selector' attribute.

   Note: Save the generated graphs to the local machine to be viewed. You cannot open images in this platform.

You can use the **GPT-5 Nano** with the following prompt:

```
For a given data set available in a CSV format, write a small and compact python code that does the following:
1. Load the CSV data to a pandas data frame assuming the first row of the CSV contains the headers of the data.
2. Save an image to disk, containing histograms of the data distribution with respect to a continuous attribute 'Age' and discrete a
3. Save an image to disk containing the correlation heatmap of the data set.
4. Identify the top 5 features with highest absolute correlation with the attribute `Selector`.
```

You can expect the generated code to be similar to the one shared below.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the CSV data into a pandas data frame
data = pd.read_csv('data.csv')
# Save histograms of data distribution for 'Age', 'Gender', and 'Selector'
plt.figure()
data['Age'].hist()
plt.savefig('age_histogram.png')
plt.figure()
data['Gender'].hist()
plt.savefig('gender_histogram.png')
plt.figure()
data['Selector'].hist()
plt.savefig('selector_histogram.png')
# Save correlation heatmap of the data set
plt.figure()
corr = data.corr()
sns.heatmap(corr, annot=True)
plt.savefig('correlation_heatmap.png')
# Identify top 5 features with highest absolute correlation with 'Selector'
correlation_with_selector = corr['Selector'].abs().sort_values(ascending=False)
top_5_features = correlation_with_selector[1:6]
# Exclude 'Selector' itself
print(top_5_features)
```

You can edit the URL, adjust a few lines of the code to make the output a little more usable, and the final code should look as shown below.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the CSV data into a pandas data frame
data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMSkillsNetwork-AI0273EN-SkillsNetwork/labs/
# Save histograms of data distribution for 'Age', 'Gender', and 'Selector'
plt.figure()
data['Age'].hist()
plt.savefig('age_histogram.png')
plt.figure()
data['Gender'].hist()
plt.savefig('gender_histogram.png')
plt.figure()
data['Selector'].hist()
plt.savefig('selector_histogram.png')
# Save correlation heatmap of the data set
plt.figure(figsize=(12,8))
corr = data.corr()
sns.heatmap(abs(corr), annot=True)
plt.savefig('correlation_heatmap.png', bbox_inches='tight')
# Identify top 5 features with highest absolute correlation with 'Selector'
correlation_with_selector = corr['Selector'].abs().sort_values(ascending=False)
top_5_features = correlation_with_selector[1:6]
# Exclude 'Selector' itself
print(top_5_features)import argparse
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from typing import List, Tuple
def load_dataframe(csv_path: str) -> pd.DataFrame:
    try:
        return pd.read_csv(csv_path)
    except Exception as e:
        raise SystemExit(f"Error loading CSV '{csv_path}': {e}")
def plot_histograms(df: pd.DataFrame, hist_path: str) -> None:
    """Plot histograms:
    - Age distribution (overall) by Gender (if present)
    - Age distribution by Selector (if present)
    """
    sns.set(style="whitegrid")
    fig, axes = plt.subplots(1, 2, figsize=(12, 4.5))
    if 'Age' in df.columns:
        if 'Gender' in df.columns:
            sns.histplot(data=df, x='Age', hue='Gender', multiple='stack', ax=axes[0])
            axes[0].set_title('Age by Gender')
        else:
            axes[0].text(0.5, 0.5, 'Gender missing', ha='center', va='center')
            axes[0].axis('off')
        if 'Selector' in df.columns:
            sns.histplot(data=df, x='Age', hue='Selector', multiple='stack', ax=axes[1])
            axes[1].set_title('Age by Selector')
```

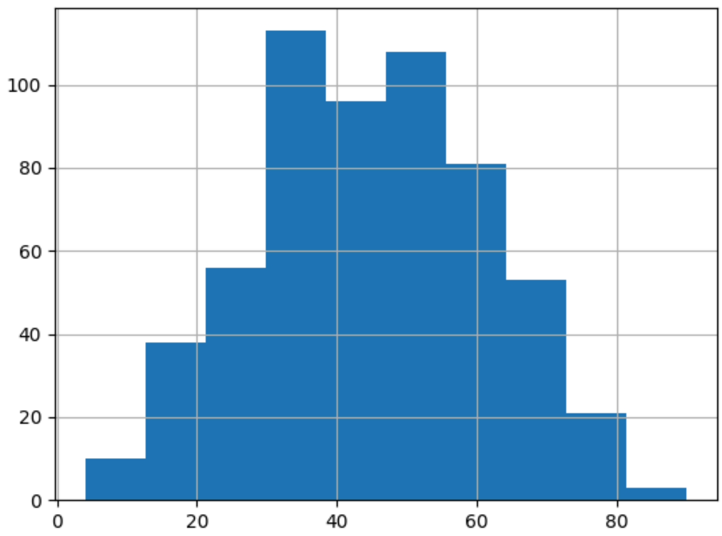
```python
        else:
            axes[1].text(0.5, 0.5, 'Selector missing', ha='center', va='center')
            axes[1].axis('off')
    else:
        for ax in axes:
            ax.text(0.5, 0.5, 'Age missing', ha='center', va='center')
            ax.axis('off')
    plt.tight_layout()
    plt.savefig(hist_path)
    plt.close(fig)
def plot_heatmap(df: pd.DataFrame, heatmap_path: str) -> None:
    """Create and save a correlation heatmap using one-hot encoded categoricals."""
    encoded = pd.get_dummies(df, drop_first=False)
    corr = encoded.corr()
    plt.figure(figsize=(12, 10))
    sns.heatmap(corr, cmap='coolwarm', center=0, linewidths=0.5, square=False)
    plt.tight_layout()
    plt.savefig(heatmap_path)
    plt.close()
def top_features_by_selector(df: pd.DataFrame, top_n: int = 5) -> List[Tuple[str, float]]:
    """Return top_n features with highest absolute correlation to 'Selector'."""
    if 'Selector' not in df.columns:
        return []
    # Numeric Selector: standard Pearson on numeric features
    if pd.api.types.is_numeric_dtype(df['Selector']):
        numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
        if 'Selector' not in numeric_cols:
            numeric_cols.append('Selector')
        sub = df[numeric_cols]
        corr = sub.corr()
        if 'Selector' in corr.columns:
            series = corr['Selector'].abs().drop(labels=['Selector'], errors='ignore')
            top = series.sort_values(ascending=False).head(top_n)
            return [(name, float(val)) for name, val in top.items()]
        return []
    else:
        # Categorical Selector: one-hot encode and compare with any Selector category
        sel = pd.get_dummies(df['Selector'], prefix='Selector')
        others = df.drop(columns=['Selector'])
        enc = pd.get_dummies(others, drop_first=True)
        if enc.shape[1] == 0 or sel.shape[1] == 0:
            return []
        sel, enc = sel.align(enc, join='left', axis=0, fill_value=0)
        combined = pd.concat([enc, sel], axis=1)
        scores: dict = {}
        for feat in enc.columns:
            max_corr = 0.0
            for s_col in sel.columns:
                c = combined[feat].corr(combined[s_col])
                if pd.notnull(c):
                    max_corr = max(max_corr, abs(float(c)))
            scores[feat] = max_corr
        top_items = sorted(scores.items(), key=lambda kv: kv[1], reverse=True)[:top_n]
        return top_items
def save_top_features(items: List[Tuple[str, float]], path: str) -> None:
    with open(path, 'w', encoding='utf-8') as f:
        for name, val in items:
            f.write(f"{name}\t{val:.6f}\n")
def main() -> None:
    parser = argparse.ArgumentParser(description="Compact CSV analysis: histograms, heatmap, and top features.")
    parser.add_argument("csv_path", help="Path to input CSV file (headers in first row)")
    parser.add_argument("hist_path", help="Path to save age distribution histograms image")
    parser.add_argument("heatmap_path", help="Path to save correlation heatmap image")
    parser.add_argument("--top_features_path", default=None, help="Optional path to save top features to file")
    args = parser.parse_args()
    df = load_dataframe(args.csv_path)
    # Histograms
    try:
        plot_histograms(df, args.hist_path)
        print(f"Saved histograms to: {args.hist_path}")
    except Exception as e:
        print(f"Warning: could not create histograms: {e}")
    # Heatmap
    try:
        plot_heatmap(df, args.heatmap_path)
        print(f"Saved heatmap to: {args.heatmap_path}")
    except Exception as e:
        print(f"Warning: could not create heatmap: {e}")
    # Top features
    feats = top_features_by_selector(df, top_n=5)
    if feats:
        print("Top features by absolute correlation with 'Selector':")
        for name, val in feats:
            print(f"  {name}: {val:.6f}")
        if args.top_features_path:
            try:
                save_top_features(feats, args.top_features_path)
                print(f"Saved top features to: {args.top_features_path}")
            except Exception as e:
                print(f"Warning: could not save top features: {e}")
    else:
        print("No top features found (check dataset).")
if __name__ == "__main__":
    main()
```

The required outputs would be as shown below.

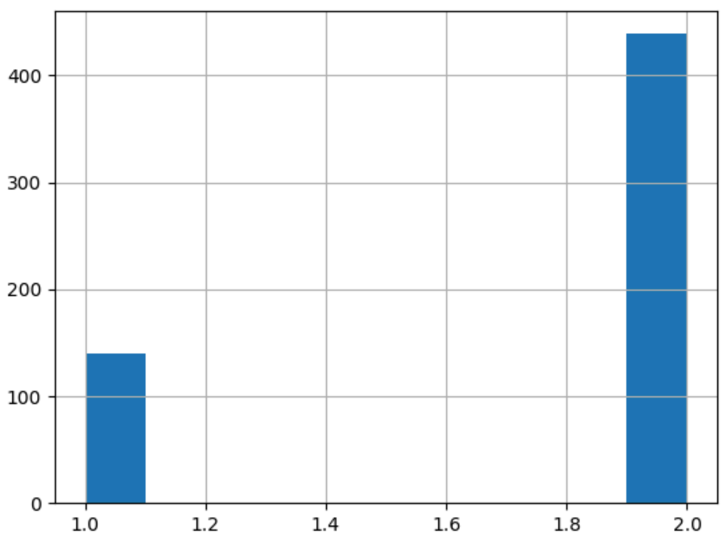
1. Terminal Output

```
theia@theia-abhishekg1:/home/project$ python3 test_file.py
Direct_Bilirubin              0.246273
Total_Bilirubin               0.220218
Alkaline_Phosphotase          0.183363
Albumin and Globulin Ratio    0.163131
Alamine_Aminotransferase      0.163117
Name: Selector, dtype: float64
```

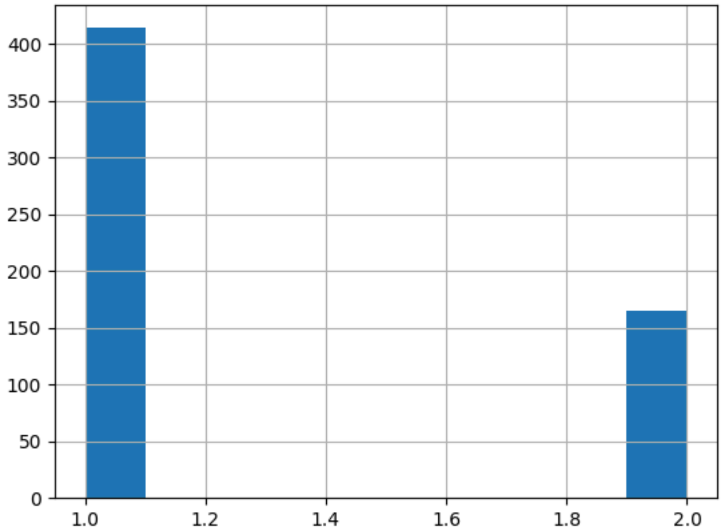
2. Age histogram



3. Gender histogram



4. Selector histogram



5. Correlation heatmap

Data Mining strategies

Data mining deals with identification of patterns in the distribution of data. From building classifiers to identifying association between different attributes, the data mining strategies can be very vast. In this task, you will use generative AI to create a classifier that will take the best 5 identified features and create a Liver Disease detection classifier. To further illustrate the power of the generative AI tools, you can include the following aspects to the code.

1. Perform standard scaling on the attributes.

2. Compare the classification accuracy of multiple classifiers on the data to identify the best one. The ones that should be used are:
   a. Logistic regression
   b. KNN
   c. Naive Bayes
   d. Decision trees
   e. Random forests
   f. Multilayer perceptron

You can create this code simply by using the following prompt.

```
Write a brief and compact python code that can perform the following tasks:
1. Perform standard scaling operation on the top 5 attributes identified in the previous code.
2. Train and test the following classifiers on the data and print a comparative table for their accuracy performance:
   a. Logistic Regression
   b. KNN
   c. Naive Bayes
   d. Decision Trees
   e. Random Forests
   f. Multi layer perceptron
```

You can expect a code to be generated, similar to the one shown below.

```
import argparse
import numpy as np
import pandas as pd
from typing import List, Tuple
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
def load_csv(path: str) -> pd.DataFrame:
    try:
        return pd.read_csv(path)
    except Exception as e:
        raise SystemExit(f"Error loading CSV '{path}': {e}")
def compute_top_features(df: pd.DataFrame, top_n: int = 5) -> List[Tuple[str, float]]:
    if 'Selector' not in df.columns:
        return []
    # Numeric Selector: use Pearson correlation on numeric features
    if pd.api.types.is_numeric_dtype(df['Selector']):
        numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
        if 'Selector' not in numeric_cols:
            numeric_cols.append('Selector')
        sub = df[numeric_cols]
        corr = sub.corr()
        if 'Selector' in corr.columns:
            s = corr['Selector'].abs().drop(labels=['Selector'], errors='ignore')
            top = s.sort_values(ascending=False).head(top_n)
            return [(name, float(val)) for name, val in top.items()]
        return []
    else:
        # Categorical Selector: encode and compare with any Selector category
        sel = pd.get_dummies(df['Selector'], prefix='Selector')
        others = df.drop(columns=['Selector'])
        enc = pd.get_dummies(others, drop_first=False)
        if enc.shape[1] == 0 or sel.shape[1] == 0:
            return []
        sel, enc = sel.align(enc, join='left', axis=0, fill_value=0)
        combined = pd.concat([enc, sel], axis=1)
        scores: dict = {}
        for feat in enc.columns:
            max_corr = 0.0
            for s_col in sel.columns:
                c = combined[feat].corr(combined[s_col])
                if pd.notnull(c):
                    max_corr = max(max_corr, abs(float(c)))
            scores[feat] = max_corr
        top_items = sorted(scores.items(), key=lambda kv: kv[1], reverse=True)[:top_n]
        return top_items
def main():
    parser = argparse.ArgumentParser(description="Scale top-5 features and evaluate classifiers.")
    parser.add_argument("csv_path", help="Path to input CSV file (headers in first row)")
    args = parser.parse_args()
    df = load_csv(args.csv_path)
    if 'Selector' not in df.columns:
        print("Selector column not found; exiting.")
        return
    # One-hot encode features (excluding target)
    X_raw = df.drop(columns=['Selector'])
    X_enc = pd.get_dummies(X_raw, drop_first=False)
    X_enc = X_enc.fillna(0.0)
    # Target encoding
    y = df['Selector'].astype(str)
    le = LabelEncoder()
    y_enc = le.fit_transform(y)
    if len(np.unique(y_enc)) < 2:
```

```
        print("Not enough classes in Selector for classification.")
        return
    # Identify and scale top-5 features
    top_feats = compute_top_features(df, top_n=5)
    top_names = [name for name, _ in top_feats]
    cols_to_scale = [c for c in top_names if c in X_enc.columns]
    if cols_to_scale:
        from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()
        X_enc[cols_to_scale] = scaler.fit_transform(X_enc[cols_to_scale])
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X_enc, y_enc, test_size=0.2, random_state=42, stratify=y_enc
    )
    models = [
        ("Logistic Regression", LogisticRegression(max_iter=1000, solver='lbfgs', multi_class='multinomial')),
        ("KNN", KNeighborsClassifier()),
        ("Naive Bayes", GaussianNB()),
        ("Decision Tree", DecisionTreeClassifier()),
        ("Random Forest", RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1)),
        ("MLP", MLPClassifier(hidden_layer_sizes=(100,), max_iter=300))
    ]
    print("Model\tAccuracy")
    for name, model in models:
        try:
            model.fit(X_train, y_train)
            preds = model.predict(X_test)
            acc = accuracy_score(y_test, preds)
            print(f"{name}\t{acc:.3f}")
        except Exception as e:
            print(f"{name}\tError: {e}")
if __name__ == "__main__":
    main()
```

As the final output, you can expect the following data frame to be printed.

	Classifier	Accuracy
0	Logistic Regression	0.620690
1	KNN	0.646552
2	Naive Bayes	0.577586
3	Decision Trees	0.603448
4	Random Forests	0.620690
5	Multi layer perceptron	0.620690

Conclusion

Congratulations on completing this lab!

You now know how to use generative AI to:

1. Perform exploratory data analysis
2. Implement data mining strategies

Author(s)

[Abhishek Gagneja]