Project Task - Week 1 : Data Science

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import gender_guesser.detector as gender
from datetime import datetime
```

STEP 1: Load datasets

```
# Load and merge the data
hospital_df = pd.read_csv(r"D:\Payal\Data Science\Capstone Project\Healthcare_Insurance_Anal
medical_df = pd.read_csv(r"D:\Payal\Data Science\Capstone Project\Healthcare_Insurance_Analy
names_df = pd.read_csv(r"D:\Payal\Data Science\Capstone Project\Healthcare_Insurance_Analysi
```

```
print(type(hospital_df), type(medical_df), type(names_df))
```

```
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.frame.DataFrame'> <class 'pandas.c
```

```
#  Merge datasets on 'Customer ID'
df = hospital_df.merge(medical_df, on='Customer ID', how='left')
df = df.merge(names_df, on='Customer ID', how='left')
```

```
print(df.columns)
```

```
Index(['Customer ID', 'year', 'month', 'date', 'children', 'charges',
       'Hospital tier', 'City tier', 'State ID', 'BMI', 'HBA1C',
       'Heart Issues', 'Any Transplants', 'Cancer history',
       'NumberOfMajorSurgeries', 'smoker', 'name'],
      dtype='object')
```

Step 2: Check missing values

```
df.isnull().sum()
```

```
Customer ID               0
year                      0
month                     0
date                      0
children                  0
charges                   0
Hospital tier             0
City tier                 0
State ID                  0
BMI                       8
HBA1C                     8
Heart Issues              8
Any Transplants           8
Cancer history            8
```

```
NumberOfMajorSurgeries    8
smoker                    8
name                      8
dtype: int64
```

STEP 3: Clean data or handle trivial values

```python
# Check percentage of '?' values
for col in df.columns:
    if df[col].dtype == 'object':
        print(f"{col}: {(df[col] == '?').mean() * 100:.2f}%")


# Remove rows with too many '?' or impute/replace if few
df.replace('?', pd.NA, inplace=True)
df.dropna(inplace=True)
```

```
Customer ID: 0.26%
year: 0.09%
month: 0.13%
Hospital tier: 0.04%
City tier: 0.04%
State ID: 0.09%
Heart Issues: 0.00%
Any Transplants: 0.00%
Cancer history: 0.00%
NumberOfMajorSurgeries: 0.00%
smoker: 0.09%
name: 0.00%
```

Step 4. Encode categorical variables

```python
# Nominal: Hospital tier, City tier, Gender (after creating), State ID (limited)

nominal_cols = ['Hospital tier', 'City tier']
df = pd.get_dummies(df, columns=nominal_cols, drop_first=True)
```

Step 5. Handle State ID (keep only R1011, R1012, R1013)

```python
df['State_ID_filtered'] = df['State ID'].apply(lambda x: x if x in ['R1011', 'R1012', 'R1013
df = pd.get_dummies(df, columns=['State_ID_filtered'], drop_first=True)
```

6. Clean NumberOfMajorSurgeries

```
# Do the numeric conversion and the missing-value replacement in a single assignment:

df['NumberOfMajorSurgeries'] = (pd.to_numeric(df['NumberOfMajorSurgeries'], errors='coerce')
      .fillna(0)                        # replace NaNs with 0
      .astype(int)                      # optional: make it an integer column
)
```

7. Calculate Age

```
# Convert month from text (e.g., 'Jul') to number

df['month'] = pd.to_datetime(df['month'], format='%b', errors='coerce').dt.month

# Handle missing or bad values in year/month/day
df['year'] = pd.to_numeric(df['year'], errors='coerce').fillna(1980).astype(int)
df['month'] = df['month'].fillna(6).astype(int)

# Handle missing day/date column
if 'date' in df.columns:
    df['day'] = pd.to_numeric(df['date'], errors='coerce').fillna(15).astype(int)
else:
    df['day'] = 15  # default day if missing

# Construct DOB and calculate Age
df['DOB'] = pd.to_datetime(dict(year=df['year'], month=df['month'], day=df['day']), errors='
df['Age'] = (pd.to_datetime('today') - df['DOB']).dt.days // 365
```

Step 8 : Gender extraction function

```
def extract_gender(name):
    name = name.lower()
    if 'mr.' in name:
        return 'Male'
    elif 'ms.' in name or 'mrs.' in name or 'miss' in name:
        return 'Female'
    else:
        return 'Unknown'

# Apply gender extraction
df['Gender'] = df['name'].apply(extract_gender)


# FOR ML - when ready for modeling

df_ml = df.copy()
df_ml = pd.get_dummies(df_ml, columns=['Gender'], drop_first=False)
```

png

Figure 1: png

png

Figure 2: png

```
print(df_ml.columns)
```

```
Index(['Customer ID', 'year', 'month', 'date', 'children', 'charges',
       'State ID', 'BMI', 'HBA1C', 'Heart Issues', 'Any Transplants',
       'Cancer history', 'NumberOfMajorSurgeries', 'smoker', 'name',
       'Hospital tier_tier - 2', 'Hospital tier_tier - 3',
       'City tier_tier - 2', 'City tier_tier - 3', 'State_ID_filtered_R1011',
       'State_ID_filtered_R1012', 'State_ID_filtered_R1013', 'day', 'DOB',
       'Age', 'Gender_Male', 'Gender_Female', 'Gender_Male'],
      dtype='object')
```

```
print(df['Gender'].value_counts())
```

```
Gender
Female    1165
Male      1160
Name: count, dtype: int64
```

Step 9. Visualize Cost Distribution

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(df['charges'], kde=True)
plt.title('Histogram of Hospitalization Cost')
plt.show()

sns.boxplot(y='charges', data=df)
plt.title('Boxplot of Hospitalization Cost')
plt.show()

sns.swarmplot(y='charges', data=df.sample(100))  # for clarity
plt.title('Swarmplot of Hospitalization Cost')
plt.show()
```

png

Figure 3: png

png

Figure 4: png

Step 10. Compare Cost by Gender and Hospital Tier

```
sns.boxplot(x='Hospital tier_tier - 2', y='charges', hue='Gender', data=df)
plt.title('Hospital Tier vs Charges by Gender')
plt.show()
```

Step 11. Radar Chart of Median Cost by Hospital Tier

```
# Check the unique values in dummy columns

print(df[['Hospital tier_tier - 2', 'Hospital tier_tier - 3']].drop_duplicates())
```

```
     Hospital tier_tier - 2  Hospital tier_tier - 3
0                      True                   False
3                     False                    True
239                   False                   False
```

```
# Safely convert to numeric (if needed)

df['Hospital tier_tier - 2'] = pd.to_numeric(df['Hospital tier_tier - 2'], errors='coerce').
df['Hospital tier_tier - 3'] = pd.to_numeric(df['Hospital tier_tier - 3'], errors='coerce').

# Create the Hospital_Tier_Label column properly

def get_tier(row):
    if row['Hospital tier_tier - 2'] == 1:
        return 'Tier 2'
    elif row['Hospital tier_tier - 3'] == 1:
        return 'Tier 3'
    else:
        return 'Tier 1'

df['Hospital_Tier_Label'] = df.apply(get_tier, axis=1)

# confirm the tier labels

print(df['Hospital_Tier_Label'].value_counts())
```

```
Hospital_Tier_Label
Tier 2    1334
Tier 3     691
Tier 1     300
```

```
Name: count, dtype: int64
```

```
#  Get Median Hospital Charges by Tier

median_costs = df.groupby('Hospital_Tier_Label')['charges'].median().reset_index()
print(median_costs)
```

```
  Hospital_Tier_Label    charges
0              Tier 1  32097.435
1              Tier 2   7168.760
2              Tier 3  10676.830
```

```
# Radar Chart

from sklearn.preprocessing import MinMaxScaler
import plotly.graph_objects as go

# Scale values
scaler = MinMaxScaler()
scaled = scaler.fit_transform(median_costs[['charges']]).flatten()

# Add back the loop closure
categories = median_costs['Hospital_Tier_Label'].tolist()
values = scaled.tolist()

categories += [categories[0]]
values += [values[0]]

# Radar plot
fig = go.Figure(
    data=go.Scatterpolar(
        r=values,
        theta=categories,
        fill='toself',
        name='Normalized Charges'
    )
)

fig.update_layout(
    title='Normalized Median Hospitalization Cost by Hospital Tier',
    polar=dict(
        radialaxis=dict(visible=True, range=[0, 1])
    ),
    showlegend=False
)

fig.show()
```

png

Figure 5: png

png

Figure 6: png

<div id="33de29cf-b3d5-4f19-b5e0-4f6e9c4e8153" class="plotly-graph-d

var gd = document.getElementById('33de29cf-b3d5-4f19-b5e0-4f6e9c4e8153'); var x = new MutationObserver(function (mutations, observer) {{ var display = window.getComputedStyle(gd).display; if (!display || display === 'none') {{ console.log([gd, 'removed!']); Plotly.purge(gd); observer.disconnect(); }} }});

// Listen for the removal of the full notebook cells var notebookContainer = gd.closest('#notebook-container'); if (notebookContainer) {{ x.observe(notebookContainer, {childList: true}); }}

// Listen for the clearing of the current output cell var outputEl = gd.closest('.output'); if (outputEl) {{ x.observe(outputEl, {childList: true}); }}

}) }; }); </script> </di

Step 12: Create a stacked bar chart (Just to show comparison instead of shape)

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.barplot(x='Hospital_Tier_Label', y='charges', data=median_costs)
plt.title("Median Hospitalization Cost by Hospital Tier")
plt.ylabel("Charges ( )")
plt.xlabel("Hospital Tier")
plt.show()
```

```
# 12. Frequency Table + Stacked Bar

freq_table = pd.crosstab(df['City tier_tier - 2'], df['Hospital tier_tier - 2'])
freq_table.plot(kind='bar', stacked=True)
plt.title('City Tier vs Hospital Tier')
plt.ylabel('Number of Patients')
plt.show()
```

Step 13. Hypothesis Testing

```
df.columns
```

```
Index(['Customer ID', 'year', 'month', 'date', 'children', 'charges',
       'State ID', 'BMI', 'HBA1C', 'Heart Issues', 'Any Transplants',
       'Cancer history', 'NumberOfMajorSurgeries', 'smoker', 'name',
       'Hospital tier_tier - 2', 'Hospital tier_tier - 3',
       'City tier_tier - 2', 'City tier_tier - 3', 'State_ID_filtered_R1011',
       'State_ID_filtered_R1012', 'State_ID_filtered_R1013', 'day', 'DOB',
       'Age', 'Gender_Male', 'Gender', 'Hospital_Tier_Label'],
      dtype='object')
```

```
from scipy.stats import f_oneway, ttest_ind, chi2_contingency

# a. ANOVA for Hospital Tier
f_oneway(df[df['Hospital tier_tier - 2']==0]['charges'],
         df[df['Hospital tier_tier - 2']==1]['charges'])
```

```
F_onewayResult(statistic=61.80716141021338, pvalue=5.747856117605796e-15)
```

```
# b. ANOVA for City Tier
f_oneway(df[df['City tier_tier - 2']==0]['charges'],
         df[df['City tier_tier - 2']==1]['charges'])
```

```
F_onewayResult(statistic=0.02171927259280913, pvalue=0.8828491424635427)
```

```
# c. The average hospitalization cost for smokers is not significantly different from the a

# Step 1: Check Unique Values in 'smoker'

print(df['smoker'].unique())
```

```
['no' 'yes']
```

```
#  Step 2: Clean the 'smoker' Column

df['smoker'] = df['smoker'].astype(str).str.strip().str.lower()

# Standardize to yes/no
df['smoker'] = df['smoker'].replace({
    'y': 'yes', '1': 'yes', 'true': 'yes',
    'n': 'no', '0': 'no', 'false': 'no'
})

# Then check:

print(df['smoker'].value_counts())
```

```
smoker
no     1839
```

```
yes     486
Name: count, dtype: int64
```

```
#  Step 3: Rerun the t-test

# Filter again with cleaned data
smokers = df[(df['smoker'] == 'yes') & (df['charges'].notna())]['charges']
nonsmokers = df[(df['smoker'] == 'no') & (df['charges'].notna())]['charges']

# Sanity check
print(f"Smokers: {len(smokers)}, Non-smokers: {len(nonsmokers)}")
```

```
Smokers: 486, Non-smokers: 1839
```

```
# Run t-test if valid
if len(smokers) > 2 and len(nonsmokers) > 2:
    from scipy.stats import ttest_ind
    result = ttest_ind(smokers, nonsmokers, equal_var=False)  # Welch's test
    print(result)
else:
    print("Still not enough data to compare smokers and non-smokers.")
```

```
TtestResult(statistic=56.316832285932286, pvalue=6.697431845033993e-238, df=582.715748991257
```

```
# d. Independence test
contingency = pd.crosstab(df['smoker'], df['Heart Issues'])
chi2_contingency(contingency)
```

```
Chi2ContingencyResult(statistic=0.08588150449910657, pvalue=0.7694797581780767, dof=1, expec
       [ 293.69032258,  192.30967742]]))
```

Project Task 2 - Week 2 : Machine Learning

1. Examine Correlation Between Predictors

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load and merge the data
hospital_df = pd.read_csv(r"D:\Payal\Data Science\Capstone Project\Healthcare_Insurance_Anal
```

png

Figure 7: png

```
medical_df = pd.read_csv(r"D:\Payal\Data Science\Capstone Project\Healthcare_Insurance_Analy
names_df = pd.read_csv(r"D:\Payal\Data Science\Capstone Project\Healthcare_Insurance_Analysi

df = hospital_df.merge(medical_df, on='Customer ID', how='left').merge(names_df, on='Custome


# Only include numeric predictors for correlation matrix
numeric_df = df.select_dtypes(include='number')
corr = numeric_df.corr()

# Plot heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

2. Build Regression Models with Pipelines and K-Fold Cross-validation

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your cleaned DataFrame
# Define features and target
target = 'charges'
X = df.drop(columns=[target])
y = df[target]

# Ensure correct dtypes (convert bools to int)
for col in X.select_dtypes(include='bool').columns:
    X[col] = X[col].astype(int)

# Only keep numeric features (avoid object/datetime)
```

```
X = X.select_dtypes(include=[np.number])


# Reset index to align X and df
df = df.reset_index(drop=True)
X = X.reset_index(drop=True)
y = y.reset_index(drop=True)




# Create stratified bins from target for balanced folds
df['bins'] = pd.qcut(y, q=5, labels=False)

# Initialize fold column
df['fold'] = -1
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for fold, (_, val_idx) in enumerate(skf.split(X, df['bins'])):
    df.loc[val_idx, 'fold'] = fold

# Drop 'bins' from features now
X['fold'] = df['fold']
y = df[target]

# ==========================
#   Ridge Regression Model
# ==========================
ridge_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('ridge', Ridge())
])

ridge_params = {
    'ridge__alpha': [0.01, 0.1, 1.0, 10.0, 100.0]
}

ridge_grid = GridSearchCV(
    estimator=ridge_pipeline,
    param_grid=ridge_params,
    cv=skf.split(X.drop(columns=['fold']), df['bins']),
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

ridge_grid.fit(X.drop(columns=['fold']), y)
best_ridge_model = ridge_grid.best_estimator_
```

```
print("  Best Ridge Params:", ridge_grid.best_params_)
print("  Ridge Best CV MSE:", -ridge_grid.best_score_)

# ================================
#   Gradient Boosting Regressor
# ================================
gb_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # optional for trees
    ('gb', GradientBoostingRegressor(random_state=42))
])

gb_params = {
    'gb__n_estimators': [100, 200],
    'gb__learning_rate': [0.05, 0.1, 0.2],
    'gb__max_depth': [3, 4, 5]
}

gb_grid = GridSearchCV(
    estimator=gb_pipeline,
    param_grid=gb_params,
    cv=skf.split(X.drop(columns=['fold']), df['bins']),
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

gb_grid.fit(X.drop(columns=['fold']), y)
best_gb_model = gb_grid.best_estimator_

print("\n  Best Gradient Boost Params:", gb_grid.best_params_)
print("  Gradient Boosting Best CV MSE:", -gb_grid.best_score_)

# ================================
#   Feature Importance & Redundancy
# ================================
# Extract feature importances from GB model
gb_model = best_gb_model.named_steps['gb']
importances = gb_model.feature_importances_

importance_df = pd.DataFrame({
    'Feature': X.drop(columns='fold').columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot top 10 features
```

png

Figure 8: png

```
plt.figure(figsize=(10,6))
sns.barplot(data=importance_df.head(10), x='Importance', y='Feature')
plt.title("Top 10 Important Features (Gradient Boosting)")
plt.tight_layout()
plt.show()

# Print redundant features (very low importance)
print("\n Redundant Features (Importance < 0.01):")
print(importance_df[importance_df['Importance'] < 0.01])
```

```
 Best Ridge Params: {'ridge__alpha': 0.01}
 Ridge Best CV MSE: 23608944.797447354
Fitting 5 folds for each of 18 candidates, totalling 90 fits

 Best Gradient Boost Params: {'gb__learning_rate': 0.05, 'gb__max_depth': 5, 'gb__n_estimat
 Gradient Boosting Best CV MSE: 5676455.087419313

 Redundant Features (Importance < 0.01):
                   Feature  Importance
0                     year    0.008287
8    Hospital tier_tier - 3    0.006853
15                     Age    0.005580
5                    HBA1C    0.003216
7    Hospital tier_tier - 2    0.003062
11  State_ID_filtered_R1011    0.002401
13  State_ID_filtered_R1013    0.001872
3                  children    0.001608
1                     month    0.001045
2                      date    0.000842
14                      day    0.000722
10       City tier_tier - 3    0.000526
16              Gender_Male    0.000305
9        City tier_tier - 2    0.000160
12  State_ID_filtered_R1012    0.000070
6    NumberOfMajorSurgeries    0.000000
```

3 and 4. Case scenario: Estimate the cost of hospitalization for Christopher, Ms. Jayna

```
# Step 1: Create her input dictionary

# Compute BMI
bmi = round(85 / (1.70 ** 2), 2)  # = 29.41

# Create input dictionary with only non-zero values
input_dict = {
    'Age': 37,
    'BMI': bmi,
    'HBA1C': 5.8,
    'children': 2,
    'NumberOfMajorSurgeries': 0,
    'Cancer history': 1,
    'Any Transplants': 0,
    'Heart Issues': 0,
    'smoker_yes': 1,
    'Gender_Male': 0,
    'State_ID_filtered_R1011': 1,
    'Hospital tier_tier - 2': 0,
    'Hospital tier_tier - 3': 0,
    'City tier_tier - 2': 0,
    'City tier_tier - 3': 0
}
```

```
# Step 2: Function to prepare input aligned with model features

def prepare_input(input_dict, model_features):
    import pandas as pd
    import numpy as np
    # Initialize row with zeros for all expected features
    df = pd.DataFrame(np.zeros((1, len(model_features))), columns=model_features)
    # Fill in values we know
    for key, value in input_dict.items():
        if key in df.columns:
            df.at[0, key] = value
    return df
```

```
# Step 3: Create Jayna's input and predict


# 1. Get model features excluding 'fold'
model_features = X.drop(columns='fold').columns

# 2. Prepare Jayna's input row aligned to model
```

png

Figure 9: png

```
jayna_input = prepare_input(input_dict, model_features)

# 3. Predict using the best model
predicted_cost = best_gb_model.predict(jayna_input)[0]

# 4. Display the result
print(f" Predicted hospitalization cost for Ms. Jayna: {predicted_cost:,.2f}")
```

```
 Predicted hospitalization cost for Ms. Jayna: 4,635.93
```

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(df['charges'], bins=50, kde=True)
plt.title('Hospitalization Charges Distribution')
plt.show()
```

Project Task - SQL is created separately in word file

Project Task - Tableau is created separately in twb file